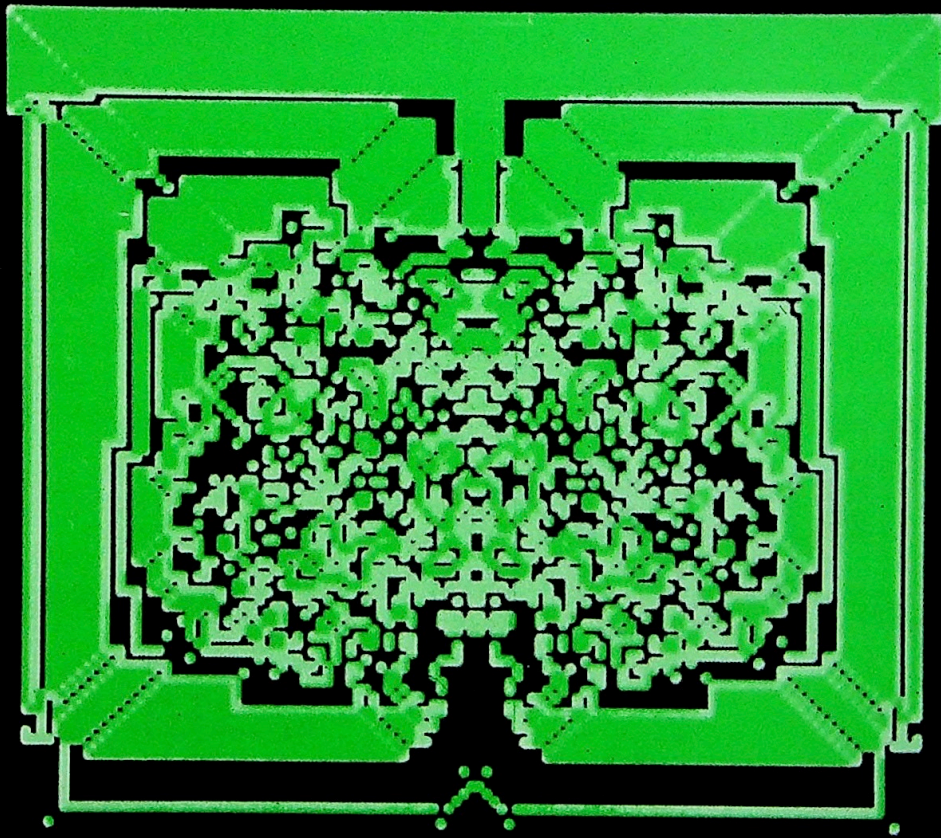


NOVEMBER 1990

Algorithm^{1.7}

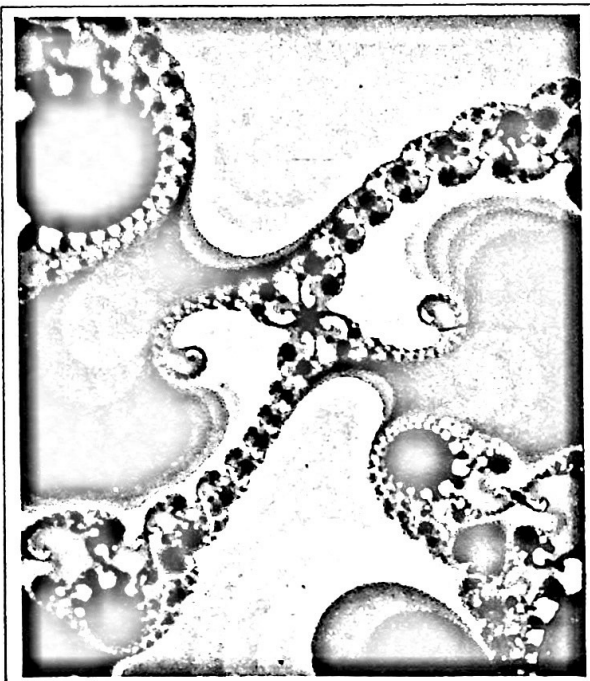
THE PERSONAL PROGRAMMING NEWSLETTER



Tur-mite patterns

Self-listing programs
Juggler sequences
Bill Gates on BASIC
A robot called Roger

**Andromeda
Research, Inc.**



Presents the Ultimate Fractal Software
For the IBM® PC and Compatibles

Fractal FX

Only one fractal program offers:

- Almost unlimited zooming capability (10^{72})
- Full screen and quadrant displays
- High-speed image generation –up to 100 times faster than the competition
- Sophisticated edge detection algorithm
- Professionally designed user interface
- Support for many high - res VGA cards (up to 1024 x 768) in both 16 and 256 color modes

Minimum Requirements:

- 8088 Microprocessor
- EGA or VGA Graphics
- Color Monitor
- 640K RAM
- Hard Disk
- MS DOS® 3.0 or later

Send \$39.95 US (check or money order) to:

Andromeda Research, Inc.
6441 Enterprise Lane
Madison, WI 53719

or call (1-800-755-3021)

VISA or Mastercard accepted
Free Shipping

EXPLOR-1

A package of programs
designed for amateur and
professional explorers alike.

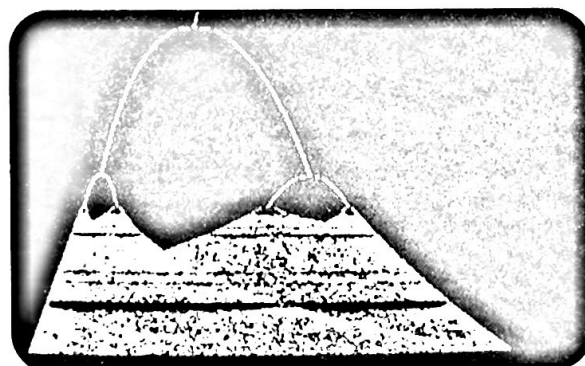
Select up to 16 programs for your own personal package: Mandelbrot and Julia sets • Chaos • Biomorphs • Computer Popcorn • Star Clusters • Wa-Tor • Party Planner • Slo-Gro • Life • Sandbox • Moiré • Wallpaper • Curves • Cyclic Space • Fractals

Ask for our free
catalog today:

TURING OMNIBUS



P.O. Box 1456,
London, Ontario
CANADA N6A 5M2



For PC and compatible computers
with CGA compatible graphics.

Algorithm

THE PERSONAL PROGRAMMING NEWSLETTER

Algostaff

Editor
A.K. Dewdney

Assistant Editors
Jennifer Connor
John O'Hagan

Managing Editor
Louis D. Magguilli

Layout & Design
Tammy Golinski

Production Services
Bogdan Graphics Inc.

ALGOLETTERS - The Readers
Serendipity Fractal
Steinhaus Patterns
The Lorentz Attraction
Going Forth Into Fractals
Experimental Exponents
Surgery on a Neural Net
Searching For Charybdis _____ 2

EASY PIECES - Michael W. Ecker
Self-shaving barbers and
self-listing programs _____ 6

COMPUTER ADVENTURES - A.K. Dewdney
Bringing tur-mites to life _____ 8

PERSONAL PROGRAMS - Clifford A. Pickover
Juggler geometry and earthworm algebra _____ 11

ALGOPUZZLE - Dennis Shasha
Assassins in the Amazon _____ 14

ALGOFAC - Susan Lammers
Programming at Microsoft-
an interview with Bill Gates _____ 16

ALGOFICTION - A.K. Dewdney
Programming Roger _____ 23

MICRO SCOPE - David Geoffrey
Grass isn't always green
If you could see what I hear _____ 26

BULLETIN BOARD
Software, games and publications from
individuals and companies _____ 28

Address all editorial correspondence to A.K. Dewdney, 362 Wortley Road, London, Ontario N6C 3S2, CANADA. Address advertising, business and all other correspondence to Louis D. Magguilli, Algorithm, P.O. Box 29237, Westmount Postal Outlet, 785 Wonderland Road S., London, Ontario N6K 1M6. Owned and published bimonthly by 854832 Ontario Inc., doing business as *Algorithm*. All rights reserved, including the right to accept or reject all editorial and advertising matter. Unsolicited materials must be accompanied by self-addressed return envelopes and postage or dollar equivalent. No part of this newsletter may be reproduced without the written permission of *Algorithm*. Copyright© 1990 by *Algorithm*. Mail subscriptions: \$29.95 (U.S.) per year; single copies \$5.00 (U.S.). Overseas add \$4.00 (U.S.) for shipping. ISSN 0843-9753. Algophone (519) 432-8042. Mailed November 1990. Second Class Mail Permit Number 8561. Postage paid in London, Ontario.

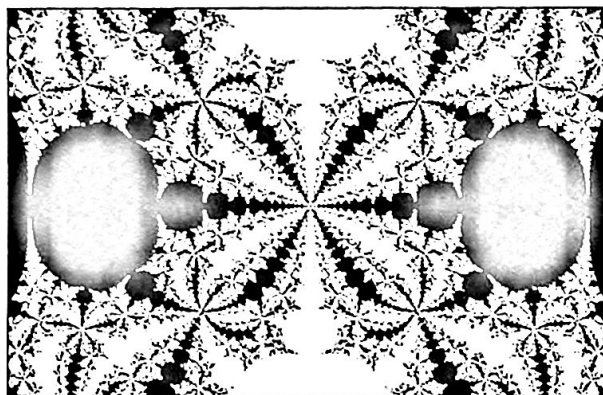
SERENDIPITY FRACTAL

Sometimes the result of a mistake can be more interesting than the correct outcome. My initial misunderstanding of Newton's method of calculating roots led me to a fractal image far more spectacular than the one that I was trying to reproduce.

Newton's method (named after, but much more ancient than, Sir Isaac) is a general technique for finding the roots of functions (e.g. $f(x) = x^n - c$). You start with a first guess x_1 . Then you calculate the next guess $x_2 = x_1 - f(x_1)/f'(x_1)$, where $f'(x)$ is the first derivative of $f(x)$. If x_2 is arbitrarily close to x_1 , then you have found a root of the function; otherwise, you calculate $x_3 = x_2 - f(x_2)/f'(x_2)$, compare x_3 to x_2 , and keep going until you have a sufficiently precise result. If the function has several roots, then it may be impossible to predict in advance which one a given first guess will lead to. Lately mathematicians and computer scientists have shown increased interest in Newton's method because of this frequently "chaotic" behavior.

The function $z^n - 1$, where z is shorthand for the complex number (x, y) and 1 is shorthand for the complex number $(1, 0)$, has roots like any other function. $(1, 0)$ has two square roots, three cube roots, four fourth roots, and so on. Different starting guesses, when you repeat the formula above, eventually lead to one root or another. It is interesting to program a computer to plot guesses and their destinations as a two-dimensional graph on a display, where each point is used as a first guess and is colored according to which of the roots it eventually leads to.

When I first experimented with



the 4th roots of $(1, 0)$ my program contained an error. The resulting image is not the one I expected. But it is an intriguing and complex fractal image unlike any I have seen anywhere. It divides the plane into four symmetrical wedges, as it should. But the chaotic borders between the wedges are not confined to a narrow strip. Instead they explode in kaleidoscopic bursts that encroach deeply into what should be the predominant basins of attraction.

Interestingly, the incorrect method "works," in the sense that every starting point does eventually lead to one of the four roots. But it is not nearly as efficient as Newton's method; this is evident from the enlarged chaotic area, and from the fact that it takes my computer fifty times as long to generate this image as to generate the orthodox one.

Steven A. Jent
Grapevine, Texas

STEINHAUS PATTERNS

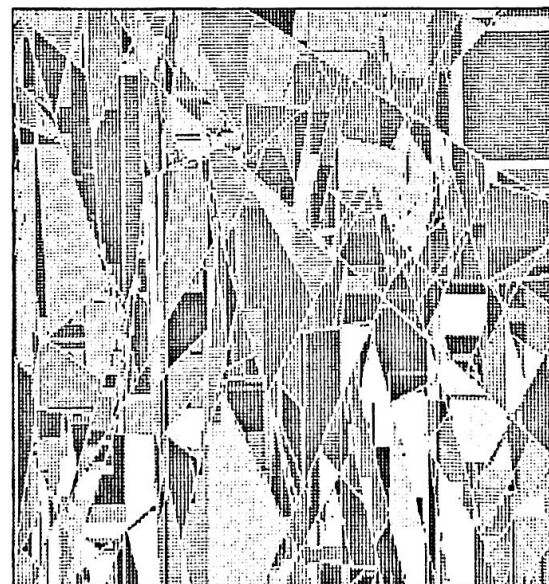
Polish mathematician Hugo Steinhaus (1887 - 1972) in 1938 published the book *Kalejdoskop matematyczny* (*Mathematical Snapshots*, see [1]), which was translated into over 10 languages. This book presents many interesting facts and ideas. Here we focus our attention on the fragment of Steinhaus' book, where a very

interesting pattern created by Nature is described: "The patterns we observe on the shore of a river when the mud has been dried up by the sun or in an earthenware jar held empty against the sunlight seem to be quite irregular; nevertheless as a rule they show right angles." Inspired by Steinhaus' pattern we develop the following algorithm to create similar patterns on the screen of a computer (for details see [2]). Start from a rectangular area determined by four white lines (the starting lines). A pseudo-random generator is used to produce points (x, y) within the considered area including the frame. If the point (x, y) lies on a horizontal (vertical) line, then from this point a vertical (horizontal) line is drawn until it meets another line. The figure illustrates the partition of the rectangular area by using not vertical and horizontal lines, but lines with different slopes. The obtained pieces were colored.

Mieczysław Szyszkowicz
Ottawa, Ontario

References

1. H. Steinhaus, *Mathematical Snapshots*, Oxford University Press, New York, 1983.
2. M. Szyszkowicz, Patterns of partitions, *Comput. & Graphics* (accepted).



THE LORENTZ ATTRACTION

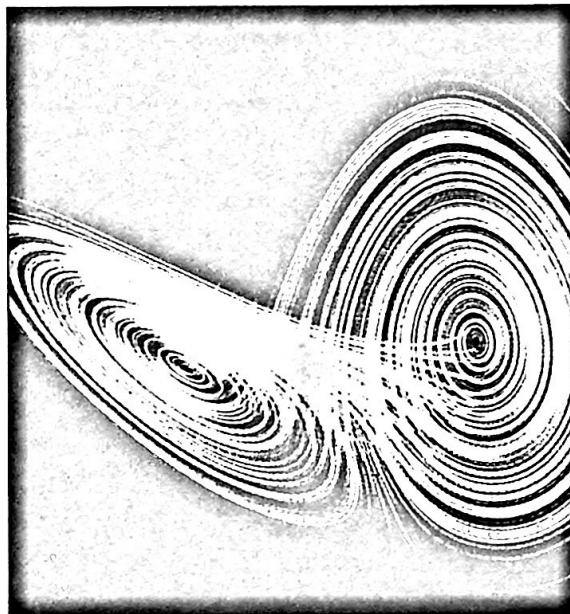
A few weeks ago I made use of your pseudo-code for the Lorentz attractor as given in *Algorithm 1.2*. Readers of one of the BBC monthly micro-computer journals had asked for examples of simple programming with interesting outputs and I could think of no better case than the Lorentz. On my Archimedes he comes across splendidly. I wrote a short program with a covering note and this has just been published with due reference to your journal.

The reason I write is to ask whether you can expand a little on the subject. My program gives the graphic output very like your Jan—Feb. illustration but what I have been (unsuccessfully) seeking is a way to arrive at a result akin to the picture above. No amount of tinkering with the trio of equations or the constants as set out in your algorithm will let me depart from the basic axes of the attractor, whereas I would like to break away as soon as ever possible! The speed and the color graphics of the Arc are very suitable for this sort of work.

Can you, or one of your contributors, give me a clue? I promise to try to get it working and then to offer it to BEEBUG as an "enhancement." Needless to say I would highlight ALGOritm once more.

P.S. I might add that I am a retired executive simply trying to keep his brain cells from Alzheimer-ing. I have great faith in computing.

D.A.A. Fagandini
6, Allyn Park
Dulwich, London SE21 8AE
England



GOING FORTH INTO FRACTALS

A friend and I had been discussing fractals for a week or two via EMail. My side of the correspondence consisted mostly of such articulations as "What's a fractal?" The August 1985 *Scientific American* with the Computer Recreations column on the Mandelbrot set appeared in the grocery store rescuing him and me!

For the first implementation of the algorithm I used Logo on my graphics engine at the time, a Commodore 64. I remember the excitement as I turned off the monitor and retired for the evening expecting to see the whole, bug-shaped black set the next morning. The next day, shocked and amazed to find only a couple of lines of color at the bottom of the monitor, I did a few calculations. It seemed that it would take that language on that computer something like 208 days to compute the image. Far too long for me.

I wrote a quick version using the built-in Commodore BASIC and timed some of that: 102 days. Serious time problems!

I searched for a faster language

that would speed things up without my actually having to learn assembly language.

I decided on Forth, that obscure language embraced by its adherents more as a religion than a computer language. I chose "Blazin' Forth Compiler," an excellent Forth written and given away to the world by that eminent if self-imposed low-profile hacker, Scott Ballantyne or SDB as he is known on many repositories of bitwise information.

I learned Forth as I went.

A year later, when the first image containing all those Mandelbrot set points appeared, it had taken slightly longer than ten days of computer time. And that was the full 480 X 480 image, counting to 1000 for each part of the set itself.

What took me so long? Learning BASIC as a first programming language, in addition to giving you a rash, cripples you when you try to learn 'real' computer languages. Even with Brodie's excellent text, I found it slow going at times. I also added lots of bells and whistles such as zooms, computing for the screen or for the 480 X 480 display of an Okimate 10 color printer, total color control, provisions for interrupting the program and continuing where you left off. Scott added a floating point package with up to 15 bytes of precision allowing filament searches without easily running out of resolution. He also coded some words (Forth functions) in assembly language to enhance speed.

Writing the documentation which includes a sample run took time as well.

Where can you get it? I gave it away to the world. You can get it from CompuServe in the CBMART library 10. It took me three hours at 300 baud to upload all of it. Some of it has been ARCD since. The search keyword is McKellar. Or see the ad

on the bulletin board in this newsletter.

Mohiedean Ghofrani (and other C-64 users), I don't know what language you're writing in on your C-64, but if it's BASIC, I'll bet Forth could speed up your programs, as well as get rid of that rash. Blazin' Forth Compiler has excellent extensions to the language giving access to the C-64 sound and graphics. There's even a turtle graphics extension that makes plotting all the wonderful graphics projects from *Algorithm* a breeze. It follows the standard in Brodie's *Starting Forth* from Prentice-Hall, Inc., Englewood Cliffs, NJ 07632, U.S.A. so you can learn it from that text. It includes documentation for the extensions. Scott says of BFCIs availability: "On CompuServe and elsewhere." In Iran you may have trouble getting it. I can send you a copy if you need and want it.

Christopher McKellar
Salt Lake City, Utah

EXPERIMENTAL EXPONENTS

The Mandelbrot iteration, $z \leftarrow z^2 + c$, yields countless fascinating images, but isn't this formula a bit boring? Generalizing the z^2 term to other integer exponents, z^n , or to real exponents, z^x , or even to complex powers, z^w , provides many new mathematical worlds to explore.

Integer Exponents

A Mandelbrot iteration of the form $z \leftarrow z^n + c$, where n is a positive integer, produces an image with $n-1$ "primary lobes." The $n=2$ "classic" Mandelbrot set has but a single primary lobe to the left of the central black area. For $n=3$ the two primary lobes are more recognizable because of symmetry. For larger n , the lobes become ever smaller and shrink around the unit circle. This suggests that the "area" of the Mandelbrot iteration $z \leftarrow z^n + c$

approaches π as n approaches infinity, even though its fractal perimeter may be infinitely long (a conjecture).

Real Exponents

In the iteration $z \leftarrow z^x + c$, the "classic" Mandelbrot set becomes a single point in an infinite, continuous real x parameter space. A "big bang" occurs for $x=1.0$, and increasing x to larger values shows a continuous "evolution" of the Mandelbrot creature, which has many surprises. One such surprise occurs when creating Julia sets using parameters from near the Mandelbrot shoreline. The image in the figure below is from the Julia set iteration $z \leftarrow z^{1.5} - 0.2$. This spiraling tree of biomorphic cells is quite interesting.

Complex Exponents

The generalization can even be extended to complex exponents using the iteration $z \leftarrow z^w + c$ with complex w . For example, the image for the Mandelbrot iteration $z \leftarrow z^{1+i} + c$ reveals a strange fractal world. Enlargements of some regions show many triangular "teeth" or "saw blades" in unusual patterns.

Computational Algorithms

So how are the values z^n , z^x and z^w , calculated? The integer powers of z can be calculated by repeated complex multiplication, or by using DeMoivre's Theorem. By substituting real x for integer n , a generalized DeMoivre's Theorem provides a way to calculate z^x :

$$z^x = r^x (\cos x\theta + i \sin x\theta)$$

In converting from rectangular (x, y) coordinates to polar (r, θ) coordinates, be careful to restrict the principal range of θ to the interval

$$-\pi < \theta \leq \pi,$$

by using the *ATAN2* function available in FORTRAN or C.

Real functions $\sin(x)$ or $\cos(x)$ are available in most math libraries, but real and complex exponentiation operators are not always available. The $\ln(x)$ and $\exp(x)$ functions can be used to calculate the real quantity r^x :

$$r^x = e^{x \ln(r)}$$

A similar functional form is used for complex exponents:

$$z^w = e^{w \ln(z)}$$

But how are the complex functions $\ln(z)$ and $\exp(z)$ calculated?

With the help of the polar form of a complex number, namely,

$$z = r e^{i\theta}$$

the complex natural logarithm becomes

$$\ln(z) = \ln(r) + i\theta$$

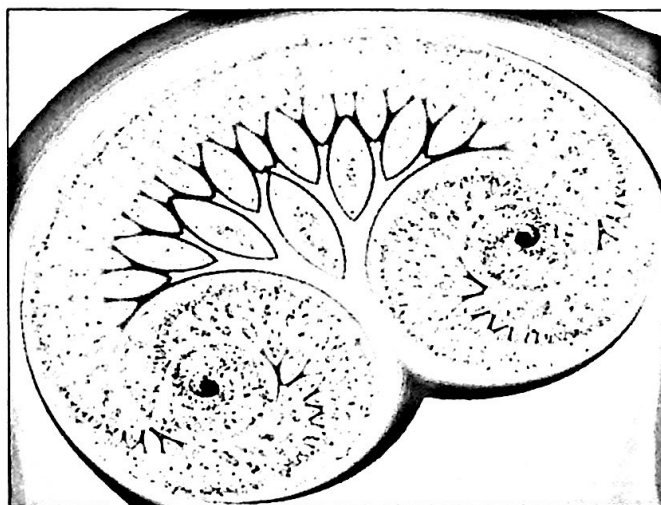
where $-\pi < \theta \leq \pi$. The complex exponential $\exp(z)$ is simplified by using Euler's formula:

$$e^z = e^{x+iy} = e^x (\cos y + i \sin y)$$

Summary

The "classic" Mandelbrot set provided countless interesting images, but this generalized approach provides even more mathematical worlds to explore with spare computer time.

Earl F. Glynn
Overland Park, Kansas



Algoletters are clearly written descriptions of computing ideas not exceeding 200 words in length.

SURGERY ON A NEURAL NET

Thank you for the postcard confirming the errors in the Neural Net pseudocode that appeared in Rietman/Frye's article in *Algorithm*. Also thanks for the errata box that appears in the recent issue. I spoke with John Lauer on the phone, who states that his suggestion was actually printed backwards in the errata box. In fact, the code should be $\text{sech}(h)^2$.

In any case, my net works beautifully now. Coded in Forth, it is available to anyone who cares to download it from the Dwight School BBS (212) 879-2625, 1200/2400 baud, 24 hrs, no fees. The file is entitled NEURNET.LZH and includes a file of trained weights which users can load into their nets, thus skipping the lengthy training period. Users will need the unarchiving program LHARC to extract the five files included in NEURNET.LZH

Thanks for producing a stimulating publication.

Nick Didkovsky
New York, NY

SEARCHING FOR CHARYBDIS

Some fathers buy their sons train sets. I bought my daughters an 8bit BBC-B microcomputer, but they quickly became bored with "Space Invaders" games and saw no point in programming trivial exercises to which they knew the answer before they started to program. So they went back to their cello and campanology, leaving me to play with the microcomputer and what they called "Mummies Mangled Brats." The micro has 8 colors at low resolution or 4 at high, which allows me to use an area of 256 by 256 pixels on the screen.

I found many exciting places. I particularly like the place I call "Charybdis, the source of the whirlpools", which is centered at $x = -1.9404811111$, $y = 0.0000161901$ with a width of 0.0000000329. See screen dump below.

I am always keen to see other interesting places, so I get frustrated when elegant pictures are published, like that one on the front of *Algorithm* 1.1, which give neither co-ordinates nor magnification nor resolution of the printer so I cannot easily reproduce them for myself.

Recently a UK micro magazine published a new program which promised me superfast plots with 35 colors without my having to buy a graphics extension ROM. (Most of my salary is used up putting my daughters through college). Unfortunately, this magazine neither published co-ordinates nor discussed the problems which might arise due to the use of integer arithmetic, but

I liked the colors, so I looked for Charybdis.

I could almost see the outline of Charybdis, when the next zoom suddenly caused wildly jagged streaks to appear, and after the next one the whole screen became a serene blue, all over! No Charybdis, just a calm sea. Nothing in the program article to suggest why this might be.

Please when you write an article for *Algorithm* about the Mandelbrot Set include enough details to enable readers to make similar plots, and warn them of any limits in screen resolution or precision of the arithmetic.

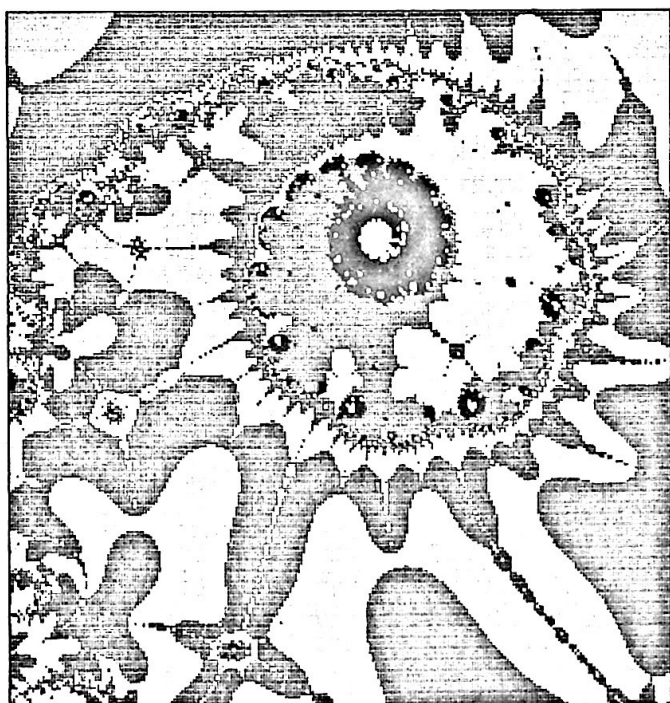
Kate Crennell
Didcot, England

SOMEBODY YOU KNOW...

... does not know about *Algorithm* yet and would like to become a regular reader. Tell him or her about the three regular recreational programming features that we include in each issue, as well as the four Algos: letters, puzzles, fact and fiction. Micro Scope reviews not only interesting professional products but fascinating programs written by other readers. The Bulletin Board lists a multitude of weird and wonderful opportunities as well, from organizations to software.

Tell your friend that we are planning to introduce (soon) a new feature in which readers write articles that describe their own programming adventures. There's more--so much more--to come, but we need your help. If the readership expands, so can *Algorithm*!

A.K. Dewdney
Editor



WE'VE MOVED!

Algorithm: editorial office
362 Wortley Road
London, Ontario
CANADA N6C 3S2

Self-shaving barbers and self-listing programs

Sal is a barber in the town of Seville. He shaves all the barbers of Seville who do not shave themselves. He shaves nobody else. The question is this: Who shaves Sal?

It's a curious riddle, for Sal must either shave himself or not shave himself. If he does not shave himself, that makes him a barber of Seville who does not shave himself and therefore one of his own best customers. But this means that he does shave himself after all.

On the other hand, if Sal does shave himself he is one of the barbers of Seville who shaves himself. But he does not shave such barbers and this means that he does not shave himself after all.

In summary, we have the classic paradox of Epimenides, also known as the Russell paradox: Sal shaves himself if and only if he doesn't!

This little chestnut has plenty of variants and more ramifications for mathematical logic than I care to deal with now. You may, however, be familiar with the version that is based on sentences: "This sentence is false." You can explore this one for yourself. Is it true if and only if it is false?

Because of their ambivalence, sentences that refer to themselves lead to assorted paradoxes and serve as a great source of humor. I recently made the following announcement to one of my classes.

"Speakers should never repeat themselves. Let me say that again. Speakers should never repeat themselves."

Not all self-referencing statements are intended as humor, paradox or puzzle. The oft-quoted Disraeli once observed with perfect seriousness: "What we learn from history is that we do not learn from history." Anybody over the age of thirty knows he wasn't kidding.

COMPUTERS AND SELF-REFERENCE

This is all good fun, you may be thinking, but what does it have to do with algorithms and computing?

First of all, if you use any application software, particularly a word processor, you have already encountered the potential for similar logical mayhem. Your keyboard probably has a number of supplementary keys such as F1 through F10 or F12, CTRL, ALT, etc. Wouldn't it be simpler to type in the command as a word rather than memorize which special key to press?

Wait! Don't dash a letter off to me just yet! My question was rhetorical. You see, the person who programmed the word processing program was aware of the pitfalls of self-reference. Let's suppose, for example, that you are running your word-processing program and you want to center some text without having to know which function-key to use. Why can't you just write "center" and have the word-processor know it?

The answer, of course, is that the word processor would treat your word as just another word to be processed. Lacking natural or artificial intelligence, the program has no way of distinguishing between text words and command words. Moreover, if the program were instructed always to interpret certain words as commands, those same words would be forbidden as parts of any document you might be preparing!

The solution is to have function keys. Thus, to center text while using the program called Total Word, I press F9. Word Perfect and other packages all have their own versions of this command. The point is, there is an ambivalence or dichotomy: if you think of the word

processor input as the object language, then the special function keys constitute a second language, a metalanguage that applies to language.

It is when we blur the distinction between the two levels of language and metalanguage that we encounter difficulties. The sentence, "This sentence is false" gives problems because language and metalanguage are indistinguishable. But now that I have set the proper philosophical tone, we are ready for some recreational computing.

SELF REFERENCE AND SELF-LISTING

A reader of *REC*, the newsletter of Recreational and Educational Computing, recently sent me a sentence that he had conjured up after five hours on his computer. It went like this:

"This sentence in this issue of *REC* Newsletter contains ____ a's, ____ b's . . . , ____ z's."

In this sentence, the blanks were filled by the appropriate numbers, each one spelled out, such as "twelve" for 12. To find out what numbers must go in the blanks is not an easy matter. After all, each time you guess a new number for one of the blanks, some of the other numbers must also change. Sentences of this type are called pangrams.

My challenge to readers is a bit less ambitious. Consider the following mystery sentence: This sentence contains exactly _____ characters.

Spaces and punctuation symbols count as characters, right up to the final period. Your mission is to write a program that will test the appropriate range of possible numbers until it comes up with a correct answer. The blank is

replaced by a number which, when spelled out, brings the total number of characters to exactly that amount. The sentence becomes self-referencing in a modest way. Readers may note that while the puzzle can easily be solved with pencil and paper, it is an enjoyable exercise to develop a program that will solve it. Moreover, other sentences can be substituted, sentences such as the following: This self-referential string contains _____ characters.

I will not spell out an algorithm for such a program. Instead, I will give only the following hints. Include DATA statements for the names of the numbers within an appropriate range; work out the counts for each of the characters in the unchanging portion of the sentence and be prepared to add these to the counts associated with each of the number names stored in the DATA statements; build a loop that systematically goes through the possibilities for the guessed number; test for equality between the guess and the actual sum of characters in the resulting sentence. Bear in mind that an arbitrary version of this problem should not, *a priori*, be assumed to have a solution. Think it through for yourself. Must there always be a solution to such a puzzle?

Some of you find Easy Pieces just a little too easy at times. Here is a non-trivial conundrum involving self-listing programs.

There is a BASIC program that consists of a single line:

1 LIST

This program has the interesting property that it has the same "output" whether you list it or run it. In other words, no matter whether you type LIST or RUN, what you see on the screen is

1 LIST

Can you write a PC program, preferably in BASIC, whose output

on executing is precisely the same as its source code as listed? You will not be allowed to use DATA statements or the LIST command. Such programs exist. I await your responses, not only to this challenge, but to the self-counting sentence problem. Send your replies, whether on paper or MSDOS disk, to me, at the address below.

READER SOLUTIONS

Of all the challenges posed in this column so far, the one I issued in *Algorithm* 1.3 (and clarified in 1.5) has generated the largest number of responses. The problem was to swap the values of two variables A and B without using a temporary third variable, a built-in swap command, or memory or utilities of any kind. You had just the two variables, plus the built-in arithmetic and logical operators with which to work.

The first solver, as I mentioned previously, was James L. Crum of Louisville, Oh. Since then I have received solutions from the following bright lights: Harry J. Smith of Saratoga, Calif.; Dr. Martin Dace of London, England; Carl C. Miller of Chippewa Falls, Wis.; Mark J. Smith of Greenfield, Mass.; Manuele Levati and Andrea Far, both of Parma, Italy; Sally Frazza of Philadelphia, Penn.; Richard Leibowitz of Arlington, Mass.; Russ Long of Media, Penn.; Shay Walters of Columbia, S.C.; Peter Hicks of Victoria, Australia; Marc Hawley of Mt. Vernon, In.; J.H. Smalley of Boulder, Colo.; Allen J. Zimmermann of New York, N.Y.; Daniel Wold of Santa Maria, Calif.; K.D. Miller Jr. of Houston, Texas; Tom Metcalf of Sacramento, Calif.; Clark Hugh Stiles of Grand Rapids, Mich.; and Mohiedean Ghofrani of Tehran, Iran.

The most straightforward solution used the following idea:

A = A+B

B = A - B

A = A - B

About half the solvers recognized that limitations in arithmetic precision and overflow might interfere with a swap of this form. Other variations that used multiplication and division had the additional problem of division by zero to worry about. Using Boolean logic, Zimmermann and others circumvented these concerns.

def *fix* (x, y) = (x and not y) or (not x and y)

let a = *fix* (a, b)

let b = *fix* (a, b)

let a = *fix* (a, b)

Mathematical types will recognize the function *fix* as the difference of two sets. Computer science types may prefer the exclusive-or operation used in assembly language programming. This approach was suggested by Far, Smith and Walters.

Harry J. Smith, incidentally, gets the palm for the largest number of distinct solutions, namely five. As the author of the shareware program VPCALC, he might be expected to know about such matters but congratulations anyway, Harry!

Miller asked whether such picky problems can serve a useful purpose. Hicks, an assembly language programmer, affirmed that such techniques used to be important when memory was at a premium. Even if memory is cheap and plentiful, similar tricks are occasionally useful for shoehorning code into tight spaces. In any event, why do we climb mountains?

Readers should feel free to write me at the following address. If you wish a reply, please be sure to include a self-addressed large (#10) envelope and (if you live in the United States) with sufficient postage on it. Until next time, may all your programming pieces be easy!

Dr. Michael Ecker, Easy Pieces
Algorithm, 909 Violet Terrace
Clarks Summit, PA 18411
USA

COMPUTER ADVENTURES

by A.K. Dewdney

Bringing tur-mites to life

Complex as they may be, the network of tunnels made by termites in earth or home are nothing compared to the patterns made by the creatures called tur-mites. A compound of "Turing" and "termite," these cybernetic creatures prowl computer screens at a million times the speed of their earth-bound cousins. They bobble, dither, wobble, and weave, all at lightspeed. Not only do their busy ways often produce amazing graphics, but they are great fun to watch. How fortunate that bringing them to life requires so little programming effort!

I first described tur-mites in the computer recreations column of *Scientific American* magazine in September, 1989. But there was little room to discuss their antics more fully or to give precise algorithmic directions for creating them. This article will rectify that.

Invented by Greg Turk while still a computer science graduate student at the University of North Carolina, tur-mites are actually two-dimensional Turing machines. Ordinary Turing machines, named after Alan Turing, the British mathematician, are essentially one-dimensional. They capture in formal terms everything that we mean by the word "computation." A Turing machine, which consists of a conceptual black box and an infinitely long tape (see the illustration below), can do anything a modern

computer can do. It just takes a lot longer to do it.

An ordinary Turing machine is like other machines in having states. In each state, it carries out actions that are characteristic of that state. The tape consists of squares, each of which contains a certain symbol from some finite set of symbols, such as {blank, 0, 1}. A Turing machine goes from state to state and moves from square to square, cycle after cycle. A cycle begins when it reads the symbol written in the square it currently scans, writes a new symbol there, moves to a new square, then enters a new state. The last three actions are determined completely by two things: what state the machine is in and what symbol it happens to read in the current square.

Most writers prefer to think of the machine remaining in a fixed location while the tape moves back and forth. But it is clearly equivalent to think of the Turing machine moving while the tape remains fixed. It happens to be the key conceptual insight in extending the Turing machines to tur-mites. Before we make the transition, however, it might be enjoyable to explore a one-dimensional computation.

In the table below, I have laid out the rules for a particular Turing machine that starts with a blank tape and ends with six 1s in a row before halting:

	1	bk
A	1LC	1RB
B	1RB	1LA
C	—	1LB

Each row of the table tells the machine what to do in each of its possible states A, B, or C. Thus in state A, if the machine reads a 1

it must write a 1 in the same square, move one square to the left (L) along the tape, then enters state C. But if in state A it reads a blank (bk), it writes a 1, moves one square to the right (R) along the tape, and enter state B. And so with the other two states that the Turing machine might be in, except that in state C, if the machine reads a 1, the corresponding table entry is blank. Absence of an instruction is tantamount to halting.

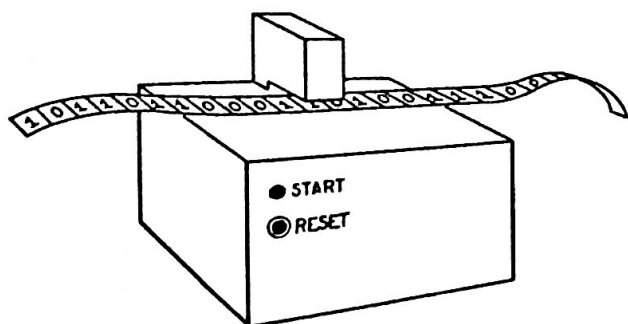
Our two-dimensional Turing machines do not necessarily halt, but an intriguing tale hangs thereby, one that I will tell a little later.

TUR-MITES EMERGE

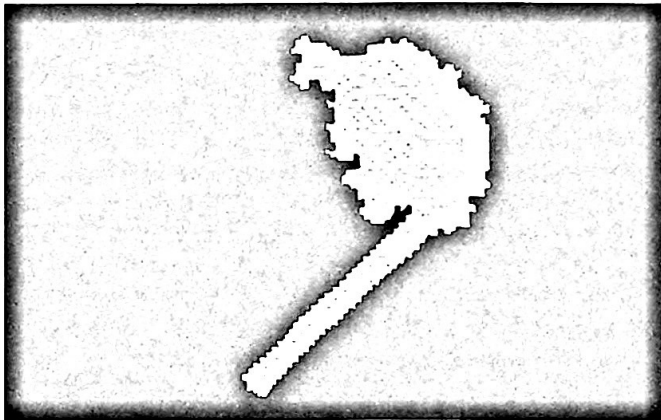
A tur-mite is a two-dimensional Turing machine with a somewhat more colorful set of symbols that we will denote by w for white, b for black and r for red. They can be changed at will to cyan, magenta or other available flavors. Our tur-mites have states, just as (one-dimensional) Turing machines do. But instead of two directions, tur-mites have four: right (R), left (L), forward (F) and back (B). The major difference between these directions and the R and L employed by the one-dimensional Turing machine is a relative matter. The latter are absolute.

The direction that a tur-mite will move next is always taken relative to its current direction. Rather than think of it as a black box with read/write head poised over a square on an infinite plane, I have chosen to think of it as a beetle like creature whose possible behaviors are worth describing.

A tur-mite carries an internalized table that is similar to the one we just examined for one-dimensional Turing machines. Moreover, a tur-mite goes through precisely the same cycle; its current state and the symbol (color) of



A Standard Turing Machine



A tur-mite makes tracks

the square it currently occupies determine what it will do next. It will first give a new color to the square it occupies, then it will move to a new square. Its current orientation will determine which way R, L, F, and B happen to lie. Its motion will be a rotation in the direction it must next move, followed by an advance of one square in that direction. In its new square it will change to a new state, the last phase of its cycle of operations. Then it is ready for a new cycle.

The following table gives the complete behavior of the tur-mite that produced the pattern on the cover of this issue of *Algorithm*. The colors are black (b), red (r), green (g), yellow (y).

	b	r	g	y
A	rLA	gRA	yRA	bLA

THE ONE-STATE TUR-MITES

Amazingly enough, this tur-mite has only one state. How much more might be expected from tur-mites with more states! Turk has made near-exhaustive exploration of one-state tur-mites. He writes, "I find it interesting that [they] generate so many remarkable patterns without any variation of states to give the

read/write heads a 'memory'... I find it interesting what these machines can do despite the limitation that they cannot carry around information in a current state."

Turk has developed a notation for these machines. It consists of a single string of

symbols in which much is implicit. Consider for example, the tur-mite whose string is RL2. The number of direction symbols such as R, L, F, and B that appear in the string equals the number of colors. Each such symbol corresponds to a unique color (arbitrary but fixed) and the tur-mite moves in that direction when it processes a square of that color. The tur-mite string is read left-to-right until a number is encountered. The color it paints the current square (before moving) is the color associated with the next position in the sequence. The tur-mite specified by the string RL2 has only two colors to deal with. Suppose they are black and red, taken in that order. If it happens to be on a black square, it colors it red and then turns right. If it comes to a red square, the tur-mite colors it black and then turns left. The 2 here means that the color to paint the cell where it turns left is the one associated with the position two symbols back.

In this notation, the string for the cover tur-mite would be LRRL4. Turk has listed some of his more fascinating finds and I will come to these in a moment. In the meantime, there is an algorithm to consider, one that enables us to program any one-state tur-mite automatically.

A tur-mite's main cycle

```

pi ← 50 / initial
pj ← 100 / position
dir ← 0 / initial direction
repeat
  col ← color at (pi, pj)
  turn ← turntab (col)
  col ← nextcolor (col)
  give col to pixel at (pi, pj)
  dir ← (dir + turn) mod 4
  if dir = 0 then pj ← pj + 1
  dir = 1 then pi ← pi - 1
  dir = 2 then pj ← pj - 1
  dir = 3 then pi ← pi + 1
until keystroke

```

HOW TO PROGRAM A TUR-MITE

The user of the program I call TURMITE inputs an array of numbers called *nextcolor*. This is an array of self-pointers, so to speak. Each entry of the array is itself an index, usually the next position in the array but also possibly the index of an earlier position.

```

input ncolors
for k ← 1 to ncolors
  input nextcolor (k)

```

The usual arena in which the tur-mites perform is uniformly colored. If the colors are numbered from 1 to *ncolors*, one of these colors must be selected and the screen (here assumed to be 100 rows by 200 columns) is painted that color.

```

input firstcolor
for i ← 1 to 100
  for j ← 1 to 200
    firstcolor (i, j)

```

The final stage of input involves the array called *turntab*, the sequence of turns that the tur-mite must make for each color that it might encounter.

```

for k ← 1 to ncolors
  input turntab (k)

```

Now begins the main cycle. It operates on the color of the tur-

mite's current position (pi , pj) to produce a new color at that position and a new position for the tur-mite to move to. The variable *dir* encodes the tur-mite's current direction and the variable *turn* specifies the tur-mite's new direction. The four directions are represented by the numbers 0 to 3, taken modulo 4. The turns that a tur-mite may take amount to additions modulo 4 according to the following table:

L	3
R	1
F	0
B	2

The rest of the algorithm is shown in the box on the previous page.

In all this I have not specified an array for the screen. In this case, where we want to use every pixel (because of the intricate patterns involved), it is faster and simpler to use the screen itself. In many program languages with screen graphics there is a way to access the color at any location, say (pi , pj), on the screen. I will indicate this by the following command.

col = *pscreen* (pi , pj)

This is the first command within the repeat loop. The variable *col* also acts as an index into the arrays *turntab* and *nextcolor*. From these TURMITE obtains the turn to be performed and the color to paint the current square before moving on. The current direction *dir* is altered by *turn* and the subsequent if-statements (which are probably better handled by a case statement) compute the tur-mite's next position by incrementing or decrementing one of pi or pj accordingly.

Admittedly, it is not as easy to type in the sequences

2, 3, 4, 1 and 3, 1, 1, 3

as it is to type LRRLR, but simple programs are the target in this

enterprise. Readers with more developed programming skills may easily add such a bell or whistle by taking the string as input, then unpacking it and, perhaps, even setting up the arrays *nextcolor* and *turntab* on that basis. In any event, such a facility would be handy to test the following tur-mites that Turk suggests:

RL2 meanders aimlessly around for a time then builds a path diagonally off the screen. This tur-mite's work can be seen in the illustration on page 9.

RF2 builds two binary counters, one on top of the other.

RLF1 fills up the plane with a four-sided pattern in a most eccentric manner.

RRL4 builds up a mirror symmetric pattern that is very complicated. Turk can't explain why it is symmetrical.

RFBF1 wanders aimlessly then suddenly settles down to an even growth along two rows.

THE BUSY TUR-MITE PROBLEM

For readers more attracted by theory than practice, there is a famous problem that extends very naturally into the present discussion. The busy beaver problem concerns the usual one-dimensional Turing machines: starting with a zero-filled tape, how many ones can a Turing machine write before halting?

If a Turing machine does not halt, it may write an infinite number of 1's. For machines that halt, the maximum number is known only for machines with up to four states. For $n=3$, the machine I described at the beginning of this article produces the

maximum number of 1's, six to be exact.

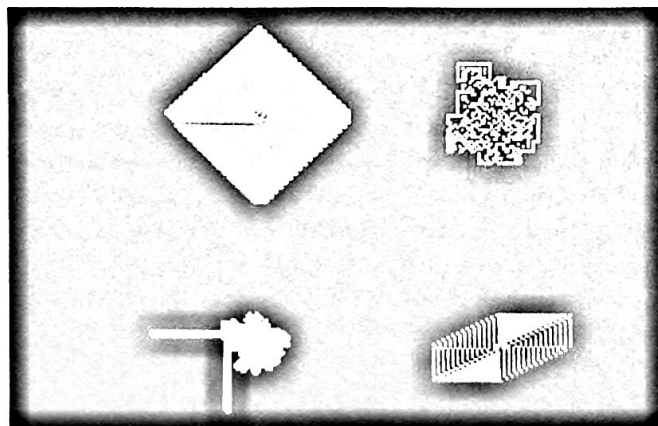
It would seem that an even greater challenge can be posed for tur-mites. The "busy tur-mite problem" would ask, for an n -state tur-mite, how many squares of an initially black space can it paint white before halting? I am open to even the most trivial advances in this (to my knowledge) completely virgin territory.

THE SCREENSAVER CONTEST

Readers are reminded that the screensaver contest closes at the end of December. Programs that keep an idle computer's screen active in the most artistic or scientifically interesting way will be awarded cash prizes as described in the Computer Adventures column in *Algorithm* 1.6. Anyone unfamiliar with this contest is urged to order *Algorithm* 1.6 as soon as possible to obtain the relevant details.

Those with interesting computer adventures to report on are invited to send me a letter at the *Algorithm* editorial address:

362 Wortley Road
London, Ontario
CANADA N6C 3S2



Some geometric tur-mites.

PERSONAL PROGRAMS by Clifford A. Pickover

Juggler geometry and earthworm algebra

Imagine a juggler who throws a ball in the air only to watch it gyrate up and down before it falls once more to his hand. What strange geometry does the path of the ball follow? Or, imagine if you will, an earthworm that can be divided in such a way that one end continually regenerates. Successive generations eventually end, where they began. Or do they? Its all a question of earthworm algebra.

Before readers wonder whether I need a vacation, let me hasten to explain. Here are two fascinating numerical recreations involving numerical sequences. The sequences are produced by two simple iterative rules: apply the rule to the current number in the sequence and you get the next one. These iterative recreations resemble the hailstone numbers (*Algorithm 1.1*) or the mathematical black holes (*Algorithm 1.5*) described by Michael Ecker in past Easy Pieces columns. In fact, the same questions come up: What is the long-term behavior of such sequences? Will they settle into a cycle? If so, what sort of cycle?

THE JUGGLER SEQUENCE

One spring morning while watching a juggler throwing brightly colored balls in the stale air of a circus tent, I thought of a simple number sequence which some friends have since spent countless hours studying. Like a ball thrown up and down by a juggler, this sequence also goes up and down, albeit in seemingly haphazard patterns. Eventually, however, the sequence seems to end up back in the jugglers hand, represented here by the number 1. If you can program your computer to calculate fractional powers (and you can), you can create a juggler sequence according to the following simple algorithm:

```
input positive integer, x
repeat
  if x is even
    then  $x \leftarrow \lfloor x^{1/2} \rfloor$ 
  else  $x \leftarrow \lfloor x^{3/2} \rfloor$ 
until  $x=1$ 
```

Starting with some positive integer as input, this iterative process continually tests the current integer x for its parity. If the number is even, it is raised to the one-half power, then truncated (the square brackets) to an integer. If the number is odd, it is raised to the three halves power before truncation.

The process continues until x again equals 1. Was I foolish to terminate the loop in this way? What if the sequence never returns to 1?

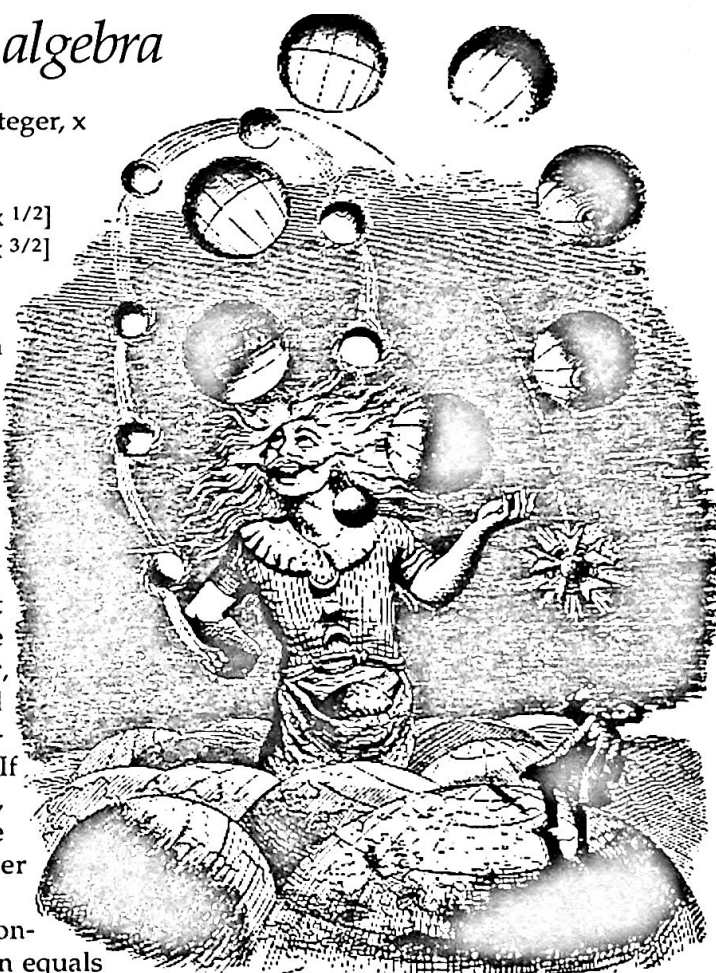
Apart from this nasty little question, the more general question of what patterns evolve is sure to delight. Consider, for example, what happens when the process goes to work on the number 3 as input:

3, 5, 11, 6, 2, 1

The basic pattern is simple enough. It rises, then it falls to 1. In general, the patterns are not quite this simple, as the next example shows:

77, 675, 17537, 2322378, 1523, 59436, 243, 3787, 2333046, 482, 21, 96, 9, 27, 140, 11, 36, 6, 2, 1

Here there are 6 ups and as many downs. The problem of deciding whether all juggler sequences eventually "fall" to 1 may be as difficult as deciding the hailstone problem. What happens



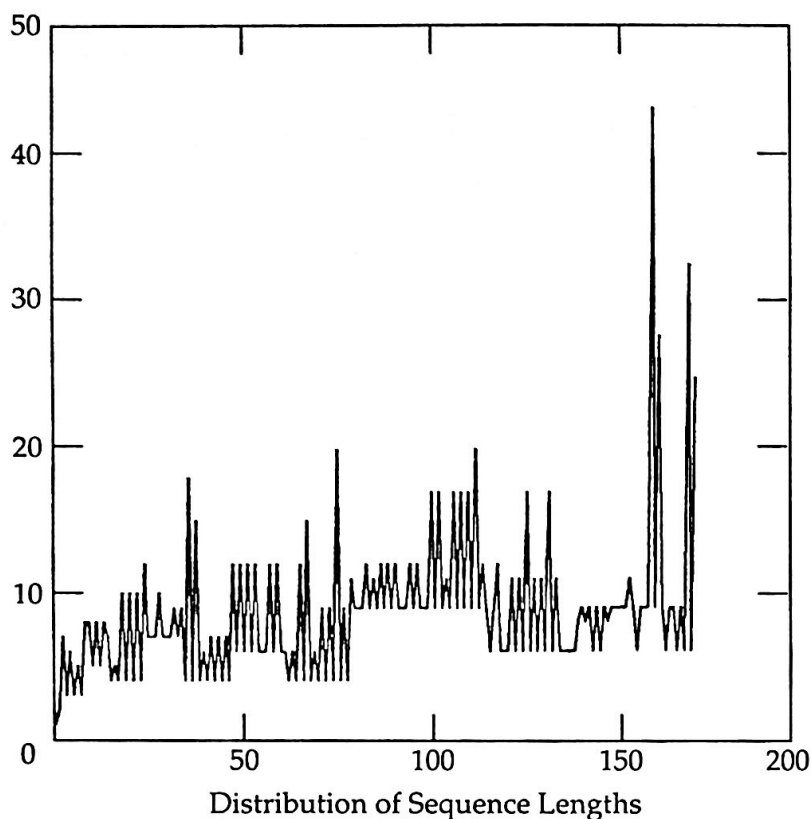
when the following rule is employed in the iteration?

```
if x is even
  then  $x \leftarrow x/2$ 
else  $x \leftarrow 3x + 1$ 
```

There is a conjecture, known as the Collatz conjecture, that hailstone sequences always fall, eventually, into the cycle 4, 2, 1, 4, ... No one has proved the Collatz conjecture yet.

Do juggler sequences seem subject to a certain law of gravity? After all, if the two power operations are performed consecutively on any number, the net result is smaller. But be warned, the numbers can get large pretty quickly. For example, the ninth member of the juggler sequence for 37 is 24906114455136.

Having conjectured that juggler sequences always fall to 1, I decided



to test my conjecture by systematically trying all the integers from 1 to 200. They all converge, ultimately, to 1. The illustration above shows all the lengths of juggler sequences from 1 to 175 in graphic form. I wonder whether any readers would like to test my conjecture beyond 200.

Other questions abound. Among my probes of the subject, I have found that sequences beginning with odd numbers are generally longer than their even cousins. In the illustration, up-and-down zig-zags in the line indicate highs and lows that consecutive even and odd integers rise to. Usually the sequence is fairly tame, reaching its maximum value in a few steps, then returning to 1 in under ten iterations. However, the numbers 37, 77, 103, 113, 115, 129, 163, 173, 183, and 193 produce fairly long sequences. Why should that be?

I have also observed that as far as the first 200 integers are concerned, only three 3-digit terminal patterns occur:

6, 2, 1; 4, 2, 1; 8, 2, 1

Why?

Readers are invited to extend

the search for longer sequences, as well as to settle the challenges I have posed here. The hardest challenge, no doubt, is to prove my conjecture. The challenge is hard to ignore because juggler sequences are so simple to define yet their universal convergence to 1 is far from guaranteed.

EARTHWORM ALGEBRA

One cool November day when I was digging in the garden, my trowel inadvertently severed an earthworm. At this time of the year many of the worms in the garden seemed to enter a breeding period, multiplying at a furious rate.

An earthworm has enormous regenerative powers, so I did not mourn for the worm. I had been taught that at least one half of the worm (and perhaps even both halves) might continue to live, depending on where the body is cut. After gazing for a few minutes at the leaf-shrouded soil, I developed a little problem called "Earthworm algebra." Like the garden scenario, it involved multiplication and

severing.

To be more accurate, Earthworm algebra is based on alternately doubling a number, then truncating it by discarding all but the last two integers. Start with any number x between 10 and 99 and watch what happens when a program, based on the following algorithm, goes to work on it:

```
input x
repeat
  x <-- trunc 2(x)
  output x
until x = a previous x
```

The algorithm is simple. Truncation of the last two digits is obtained by applying the mod function at 100, i.e., taking the remainder of x on division by 100. The algorithm ends when the variable x repeats a value taken on earlier. After that, obviously, the sequence of values will cycle endlessly and there is no point in generating more values. But how might a program detect the repetition in x -values? One way is to use an array, *val*, with 100 elements that were initially set to 0. A 1 in the k th position would mean the number k had already occurred as a value for x . Each time a new value for x is

output, the program would check the value stored in *val* (x). If it is

a 1, the program would immediately

halt. If it is a 0,

the entry is

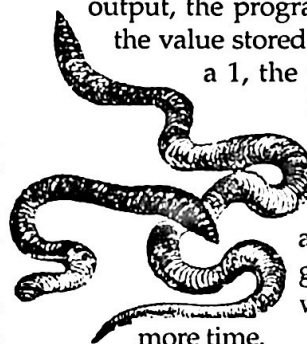
changed to a 1

and the program's main loop

would repeat one

more time.

There is a somewhat cleverer method than this which requires no auxiliary array to detect repetitions. Use two variables called *head* and *tail*, instead of x . Initially *head* and *tail* have the same starting value that x would have. New values for *head* and *tail* are computed exactly as the new value for x is computed in the algorithm above. But the same



computation is repeated one more time for *head* before the body of the loop is complete. In this way, *head* races ahead in the sequence. The cute part comes when the sequence starts repeating. Before long, the variable *head* catches up with *tail* and a simple test can be used:

until *head* = *tail*

Using the algorithm in either form, you may discover some remarkable things about earthworm algebra. My favorite phenomenon is the return of the sequence to the initial number. In general, there seem to be only three possible outcomes for these sequences: either they never return to the starting point or, if they do, they have the length 4 or 20.

The field of exploration is wide open here. What happens when the sequence is developed with other

multiplicative factors besides 2 and other moduli besides 100? A beautiful way to summarize the behavior in all cases is to draw an iteration diagram. Place each of the numbers between 0 and 99 on a sheet of paper in the form of a dot. Now apply the iteration rule you wish to investigate to each of the numbers so represented. Draw an arrow from the number to which the rule was applied to the number that resulted. Soon enough the page will fill with a confusion of arrows, but a little disentangling and redrawing will produce an elegant figure. Here are all possible earthworm sequences simultaneously visible!

As far as programming projects are concerned, the palm would certainly go to the reader who succeeded in automating the process. Imagine merely feeding in

two parameters that specified the rule, then watching the screen blossom with such a diagram!

As usual, I will be happy to hear from those readers who implement these ideas, especially from those who extend them. I can be reached by mail at the following address. Please be sure to include a self-addressed envelope and, if possible, a stamp.

Clifford Pickover, *Personal Programs*
37 Yorkshire Lane
Yorktown Heights, N.Y. 10598

Further reading

Stan Wagon,
The Collatz problem, *Mathematical Intelligencer*, 7, 1985, pp. 72-76.
Donald E. Knuth,
The Art of Computer Programming,
Vol. 2, 2nd Edition, Addison-Wesley,
Mass., 1981.

HOW TO READ ALGORITHMS

An algorithm is a generic description of a specific computation. The description does not follow the syntax of any particular programming language but it follows the overall form of a program; it consists of a sequence of informal statements. Even in our algorithms, there is some syntax, however. For example, assignment statements such as *newy* ← *a - x* use the left-arrow as a dramatic indication of how assignment statements actually work: the current values of variables that appear on the right of the arrow are combined according to the expression there (in this case, *a - x*) and the resulting number is placed in the variable (in this case *newy*). In most computer languages the left-arrow would be replaced by an equal sign (=) or colon-equals (:=).

Another feature of our algorithmic style is the use of

indentation to set off blocks of instructions. Such blocks appear inside loops or conditional statements. The following algorithm contains four indented blocks:

```
input corna, cornb
input side
for i ← 1 to 100
  for j ← 1 to 100
    x ← corna + j × side / 100
    y ← cornb + j × side / 100
    c ← int (x2 + y2)
    if c even
      then plot (i, j)
```

The first block consists of the two input statements and the for-loop. Inside the for-loop is a second for-loop comprising the second block. Inside the second for-loop is a block of four instructions that are only carried out while the inner for-loop is executing. In some programming languages, such

blocks are marked off by "begin" and "end" statements. In BASIC, a for-loop is demarked by a "for" at the beginning of the loop and a "next" at the end. The fourth block in this example consists of the plot instruction. This is indented because it is only executed when the condition (in this case, "*c* even") happens to be true.

In the informal algorithmic style, special commands can be invented and used, provided that they correspond to something a computer can do. For example, "int (*x*² + *y*²)" means, "take the integer part of the sum of *x*² and *y*²." Sometimes the shorthand is extreme: a single, informal line actually corresponds to a whole block of lines. (Readers who want to see something mildly astonishing might enjoy converting the algorithm herein to an actual program).

ALGOPUZZLE

by Dennis Shasha

Assassins in the Amazon



I turned amid the dancing shadows of the Hagglitos' campfire to survey the others in my party. Evangeline Goode, the mathematical philosopher and close friend of Dr. Ecco looked a little sad. Perhaps she wished the great solver were with us. The two University of Michigan anthropologists, Jack Raye and Peggy Rooney, were clearly fascinated by chief Sassi Nosassi, the headman of the Hagglitos, a fierce tribe that had enjoyed little contact with civilization, a tribe that still clung to its ancient ways. Finally I stared down at my own trousers, stained with river mud. What was I, Professor Scarlet, doing so far from home?

The dinner to which Sassi Nosassi had invited us consisted of grilled gar pike garnished with lizard eggs. It was surprisingly delicious and soon enough I ceased to wonder what Ecco might be doing at the moment.

Through Raye, who was proficient in the Hagglitos' tongue, the chief asked us many questions. Were we "tape recorder people" (the Hagglitos term for anthropologists) or were we prospectors? We answered that we were neither.

That seemed to puzzle the chief a great deal. "What do you want from us, then?" he asked. "Outlanders always want something."

"We just want to stay with you a few days."

Jack Raye translated Evangeline's reply for the chief, who nodded. "Very well then, welcome," he said with a slight bow.

"Beware of the assassins."

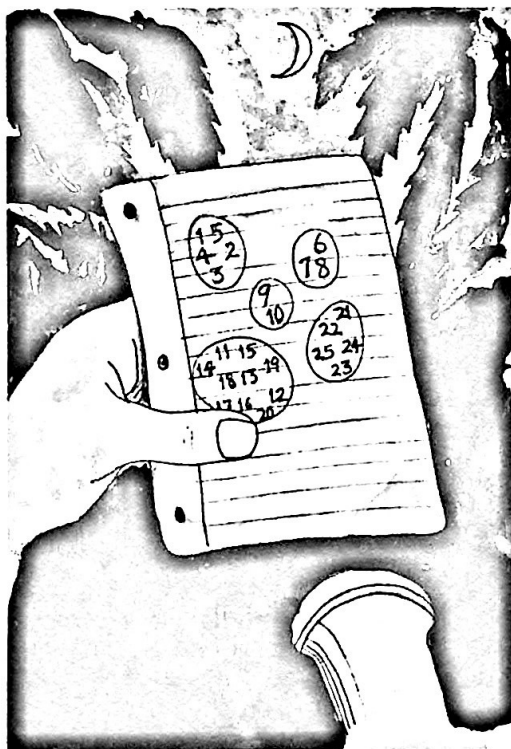
Then Sassi Nosassi stood up, a signal that the meal was over.

Evangeline and I followed the two anthropologists back to their

tents. We filed into the larger one, Raye's jungle office.

"Could you please explain the chief's warning about assassins?" said Evangeline, a distinct edge in her voice.

"Well, there's a group of village men here called assassins. Our colleague Steve Miller died at their hands last year. The fact is that



assassination is a cultural ritual for the Hagglitos. By a selection process that we don't fully understand, certain males become assassins while others do not. Each full moon, the assassins kill someone."

Evangeline and I looked for the moon simultaneously. Whew! It was a thin crescent just setting behind the trees at the west end of the clearing.

"Before his death, Miller discovered something quite interesting about the assassin cult. The non-assassins, or 'good people' in Miller's terminology, always seem to tell the truth about who is good and who is

'bad,' Miller's label for assassins. On the other hand, bad people may or may not lie. Miller discovered the presence of certain groups within Hagglitos society. He called the groups 'mutual admiration cliques' because each member of such a group would describe everyone else in the group as 'good,' in other words, as a non-assassin. He found out little else before his own untimely death at the hands of the assassins. There are apparently some good people in this village, although the chief may not be among them.

"All assassins are males that have passed the rite of manhood. In case you were afraid to ask, this means capturing a deadly curarqui snake and bringing it back to the village alive. At the moment there are 25 graduate males, so to speak, living in the village. Peggy and I have interviewed all of them. If we can discover who the good people--the non-assassins--are, we can survive by inviting them to dinner on the fine night of the full moon."

"Tribal rule?" inquired Evangeline, nervously.

"Exactly," said Peggy Rooney. "According to Hagglitos' tradition, if a host invites at least two good people to dinner on the night of the full moon, then the host is invulnerable to assassination attempts. The belief is deeply held, even by the assassins themselves. Perhaps they are afraid of avenging spirits."

"Have you interviewed the men to find out what claims they make?" I asked.

"All 25," replied Raye emphatically. "Just listen to the results: First, the initiated males all have numbers, from the chief, who is

number 1, down to the most recently initiated male who is number 25."

He bent over in the Coleman light to draw five crude circles on a sheet of paper.

"Here. These are the mutual admiration cliques that we have found. The first one contains males 1 to 5. As far as non-clique members are concerned, members of this clique would only tell me that number 18 is bad and that number 14 is good.

"The second clique we discovered consists of males 6, 7 and 8. They claim that number 13 is bad but that number 3 is good. The third clique consists of males 9 and 10 by themselves. They had nothing in particular to tell us about the other males. Now here's something interesting. The fourth group, consisting of males number 11 to 20, is almost a mutual admiration clique. The only exceptions to claims of mutual goodness involve males 18 and 19 who each claim that the other is bad. Moreover, male 15 asserts that 9 is also bad.

"Finally, the fifth clique consists of five males, 21 to 25. The only one of these males to volunteer additional information was number 22, who claimed that 9 was good while 11 was bad."

We sat up for an hour sipping a mildly intoxicating brew of fermented manioc root before slipping off to our tents.

"I'll bet Ecco would know who the bad people were by now," whispered a voice in the dark. It was Evangeline. "I'll bet I solve it before you do!"

It seemed a pity that our mutual friendship with the famed Dr. Ecco so frequently made competitors of Evangeline and me, almost like siblings vying for the attention of a father.

A little later I heard a distinct "aha" from Evangeline's tent. I could not sleep all night. Who were the assassins and who were the good

guys? Did we have enough information to decide the case for all 25 males?

Find out who should definitely be on the invitation list for the dinner of the full moon. Find out who should definitely not be invited.

I will give solutions to these separate challenges in the next installment of *Algopuzzle*. In the meantime, here is the solution to last month's tiling puzzle:

No tiling is possible for the 16 by 16 meter room. The proof uses a pigeon-hole argument that I will cast

Good guys always
tell the truth about
who is an assassin.
But an assassin may
(or may not) lie.

in the form of a Socratic dialogue. Scarlet says that he can tile the Prince's court with the 1 by 4 tiles as long as there is a 1-meter square fountain in each of the four corners. Ecco says that he cannot.

Ecco: You say you can do it?

Scarlet (pettishly): Yes, but I won't show you how.

Ecco: No matter. Suppose we number each square meter of the Prince's court with the sequence 0, 1, 2, 3 repeated over and over again. This can be done so that the numbers cycle along rows and also down columns.

Scarlet (confused): How on Earth does that affect my tiling?

Ecco: Simple! It shows you must have made a mistake. Look, are there not 64 squares labelled 0, 64 labelled 1, 64 labelled 2 and 64 labelled 3, including the squares with the fountains?

Scarlet: Yes. So what?

Ecco: Any 1-by-4 tile in your tiling, no matter in what position or orientation, must cover four squares with all four numbers on them, right?

Scarlet: Er, right!

Ecco (Inexorably): So your 63 tiles would cover 63 0s, 63 1s, 63 2s and 63 3s, would they not? But you can't do that with the four fountains in the corners because two of them will be 3s. This leaves only 62 3s for the tiles to cover.

The solution for the 10 by 10 room is simple. Create a 5-by-5 block in each corner. For example, the lower left corner could have two tiles pointing away from the corner, up and to the right. This leaves a 4 by 4 area that can be tiled either vertically or horizontally, so to speak.

Without fountains in the corners, it would be impossible to tile the 10 by 10 room. The same numbering argument used above also works here. There would be 26 squares with 1s on them but 25 tiles could not cover them.

Fred Galvin of Lawrence, Kansas proved that fountains could only be in two corners of the court. Clever readers might observe that Ecco's proof as cited above shows that the upper left and lower right corners can't both have fountains. David Wright of Oshkosh, Wis. generalized the tiling problem to show that four corners can never be tiled on a square that is $4k$ by $4k$ for any integer k .

Anyone wishing to send me solutions or correspond in general, should write me at the following address. Please use a self-addressed (and, within the US, a stamped) envelope:

Prof. Dennis Shasha, *Algopuzzle*
Courant Institute of
Mathematical Sciences
New York University
251 Mercer Street
New York, NY 10012, USA

Programming at Microsoft - an interview with Bill Gates

AS CHIEF executive officer of Microsoft, William H. (Bill) Gates is considered one of the driving forces behind today's personal computing and office automation industry. Gates started his career in computer software at a young age. Both Gates and Microsoft co-founder, Paul Allen, worked as programming consultants while attending high school in Seattle, Washington. In 1974, Gates, then an undergraduate at Harvard University, worked with Allen to develop a BASIC programming language for the first commercial microcomputer, the MITS Altair. After the successful completion of this project, the two formed Microsoft to develop and market software for the emerging microcomputer marketplace.

Gates is a native of the Seattle, Washington area and resides there today. He was born in 1955.

INTERVIEWER: You obviously have a lot of responsibilities as chief executive officer of Microsoft. Do you still program?

GATES: No, I don't. I still help design algorithms and basic approaches, and sometimes I look at code. But since I worked on the IBM PC BASIC and the Model 100, I haven't had a chance to actually create a program myself.

INTERVIEWER: What kind of role do you play in the development of programs at Microsoft?

GATES: I do two key things. One is to choose features to put into programs. To do that, you have to have a reasonable understanding of what's easy and what's not easy to do. You also have to understand what sort of product "family" strategy you're pursuing, and what's happening with the hardware.

I also work on the best way to implement that new feature, so that

it will be small and fast. For example, I wrote a memo about how to design and implement a feature we used on Excel to make the program recalculate the formulas every time the screen changes.

In the first four years of the company, there was no Microsoft program that I wasn't involved in actually writing and designing. In all those initial products, whether it was BASIC, FORTRAN, BASIC 6800, or BASIC 6502, not a line of code went out that I didn't look over. But now we have about 160 programmers, so I mostly do reviews of products and algorithms.

INTERVIEWER: What do you consider your greatest achievement ever in programming?

GATES: I'd have to say BASIC for the 8080, because of the effect it's had, and because of how appropriate it was at the time, and because we managed to get it so small. It was the original program we wrote when we decided to start Microsoft. Three of us knew that original program by heart. We got a chance to completely rewrite it one summer down in Albuquerque, and I thought we could save a few bytes and tighten things up. We just tuned the program very, very carefully, and ended up with a 4K BASIC interpreter. When you know a program that well, you feel that nobody can look at the code and say, "There's a better way to do this." That feeling's really nice, and the fact that the program was used on a lot of machines makes it an exciting program to have written.

I also feel really good about the software on the Model 100, especially about how we squeezed in a very useful small editor. I worked with Jey Suzuki, a Japanese programmer, to put that together. We had very limited time to finish the project. When you do software

that gets burned into ROM, you don't get an opportunity to make mistakes.

INTERVIEWER: What do you consider the most difficult part of computer programming?

GATES: The hardest part is deciding what the algorithms are, and then simplifying them as much as you can. It's difficult to get things down to their simplest forms. You have to simulate in your mind how the program's going to work, and you have to have a complete grasp of how the various pieces of the program work together. The finest pieces of software are those where one individual has a complete sense of exactly how the program works. To have that, you have to really love the program and concentrate on keeping it simple, to an incredible degree.

INTERVIEWER: With computers increasing so much in power and memory, are programs becoming more complex, or just sloppier? How is that affecting the way people write programs?

GATES: We're no longer in the days where every program is super well crafted. But at the heart of the programs that make it to the top, you'll find that the key internal code was done by a few people who really knew what they were doing.

It's not quite as important now to squeeze things down into a 4K memory area. You're seeing a lot more cases where people can afford to use C, instead of using assembly language. Unfortunately, many programs are so big that there is no one individual who really knows all the pieces, and so the amount of code sharing you get isn't as great. Also, the opportunity to go back and really rewrite something isn't quite as great, because there's always a new set of features that you're adding onto the same program.

The worst programs are the ones where the programmers doing the original work don't lay a solid foundation, and then they're not involved in the program in the future. Working with those programs gets to the point that I call "experimental programming." The programmers understand so little about those programs that they can't understand how changes might affect speed, for instance. They might use code that already exists, or they might not understand what dependencies will break if they change something. So they add new code, and then they run it and they say, "Oh look, it doesn't work that way." That's a very, very inefficient way to deal with a program, but a lot of projects end up exactly like that.

INTERVIEWER: In a company like Microsoft, where you have 160 programmers, how do you go about creating an environment where you can develop successful programs?

GATES: One way is to have small project teams, typically four or five people, and one of those people has to have the proven ability to really absorb a program. And when that lead person is uncertain about something, he or she should be able to discuss it with even more experienced programmers.

Part of our strategy is getting the programmers to think everything through before they go to the coding phase. Writing the design documents is crucial, because a lot of simplification comes when you see problems expressed as algorithms. They're kind of in the smallest form then, where you can see what the overlap is.

Another important element is code review, making sure that you look through the code and see if senior people can provide hints about how to do something better. And you have to review similar projects that have gone super, super

well; programmers can look at how those other people performed previously, and get ideas from the other project about how to improve their own program.

INTERVIEWER: Where do the ideas for programs come from?

GATES: Well, there's no formal process, that's for sure. At Microsoft, there's usually a brainstorming session at night or on the weekends. Everybody has a general idea, like, we want to do the world's best word processor. And we want it to allow a technical publications department to do everything they



want to do. We'll sit and talk. Well, how could we make it really fast? Could we put a drawing capability in, or could we do kerning without making the program super slow? A variety of problems will get talked through and some neat ideas will come up.

INTERVIEWER: Basically it's a group effort?

GATES: In terms of deciding what programs are going to do, a fairly large group makes suggestions. Then there's a filtering process. Eventually I'll decide which of the ideas makes sense, and I'll make sure we have champions who are personally involved in making

that product succeed. We decide to do very, very few projects because it takes incredible focusing on a lot of resources to get a product out there and set a new world class standard.

INTERVIEWER: There's a lot of talk about how large software companies find it difficult to attract talented people who can produce great software, because these mavericks are so independent that they want to work on their own. How do you attract and keep good people at Microsoft?

GATES: Great programmers are critical to create software products. But we don't believe in a *prima donna* approach, where just because somebody's good we let him not comment his code, or not communicate with other people, or impose his beliefs on everyone else.

We want people who really respect each other. I think most great programmers like to be around other great programmers. When they think up an incredible algorithm, they like having peers who can appreciate the cleverness that went into it, because when you're creating something like that and you have that model in your mind, it's a lonely thing. If you thought a process had to be complicated, and then you figure out a way to make it simpler, that makes you feel great. But you want to get some feedback from other people. Once you get a few great people, then others come.

The old rule used to be that a manager of a programmer was always a better programmer, and there were no what we called "technical inversions," where a programmer works for somebody who doesn't know how to program. We still follow that philosophy: At certain levels we've got business managers, but we don't have non-programmers really managing programming projects.

INTERVIEWER: Do you think

there are any particular rules for creating a good program?

GATES: Some people just jump in and start coding, and others think it all through before they sit down, but I think you'd find that the programmers who sit down and code at the beginning are only using that as a scratch pad. It's what's going on in their heads that's most important.

You've got to have somebody who's super smart. A great programmer thinks about the program on a constant basis, whether driving or eating. That method takes an incredible amount of mental energy.

INTERVIEWER: How would you describe your style of programming?

GATES: I like to think the whole program through at a design level before I sit down and write any of the code. And once I write the code, I like to go back and rewrite it entirely one time.

The most important part of writing a program is designing the data structures. The second most important part is breaking the various code pieces down. Until you really get in there and write it out, you don't have the keenest sense of what the common subroutines should be. The really great programs I've written have all been ones that I have thought about for a huge amount of time before I ever wrote them. I wrote a BASIC interpreter for a minicomputer in high school. I made massive mistakes in that program, and then I got to look at some other BASIC interpreters. So by the time I sat down to do Microsoft BASIC in 1975, it wasn't a question of whether I could write the program, but rather a question of whether I could squeeze it into 4K and make it super fast. I was on edge the whole time thinking, "Will this thing be fast enough? Will somebody come along and do it faster?"

I have an image in my head of this person named Norton that I met at TRW. He always showed me when I wasn't doing super well. So if I'm sloppy or lazy, I always imagine that he's going to walk up, look at the program, and tell me, "Look, here's a better way to do that." Little inefficiencies can slip into the program along the way, and if you want to really feel good about it, you've got to maintain the thought that you're not going to let that stuff creep in. That's why it's kind of painful sometimes if you have somebody else working on the project. They never code stuff exactly the way you like to see it coded. I remember when we were working on BASIC, I'd go back and recode other people's sections of code, without making any dramatic improvement. That bothers people when you go in and do that, but sometimes you just feel like you have to do it.

INTERVIEWER: When you were working with a group, were you always the design lead?

GATES: Yeah, on all the programs I've been directly involved with, I was the design lead. On the original BASIC, I scribbled out the design on pieces of paper. Paul Allen, who is a co-author, designed and implemented all the development tools.

Before I sit down to code something, most of the instructions have already run through my head. It's not all laid out perfectly, and I do find myself making changes, but all the good ideas have occurred to me before I actually write the program. And if there is a bug in the thing, I feel pretty bad, because if there's one bug, it says your mental simulation is imperfect. And once your mental simulation is imperfect, there might be thousands of bugs in the program. I really hate it when I watch some people program and I don't see them thinking.

One of the most fun programming experiences I ever had was when we were doing BASIC. I had done the 8080 BASIC, and then I had about two weeks allocated to work with Mark Chamberlain on the 6809 version of BASIC. I read the instruction set at the start of those two weeks, and I wrote about three or four programs. And I looked at some other programs to see how people used the instruction set. It was great fun to take a problem I understood and map it onto this new instruction set, and see how tightly we could put the thing together.

Programs today get very fat; the enhancements tend to slow the program down because people put in special checks. When they want to add some feature, they'll just stick in these checks without thinking about how they might slow the thing down. You have to have a programmer who knows the program inside out to guard against that. In the case of our BASIC, after I and the other original people moved away from it, we went through a period of about three years when we didn't do anything innovative. It's only in the last year and a half that we've gotten people in who feel total ownership and total understanding of our BASIC and can say, "Oh, yeah, putting subroutines in and getting rid of line numbers is easy." We've always had those goals, but until you've got that person who knows how to go into the middle of the parts or the statement analyzer and not just tack things on, you don't feel comfortable fiddling with it.

It's true that we're going to allow programs to be a little fatter than they have been. But in terms of speed, it's just laziness not to allow something to be as fast as possible, because users, even though they might not be able to say so explicitly, notice programs that are

really, really fast. In the most successful programs, the speed of execution is just wonderful.

INTERVIEWER: How do you decide on the trade-offs between speed and performance?

GATES: Sometimes it's a trade-off between adding features and executing really fast, but there are ways of having lots and lots of features and still making things fast. Basically you want to decide what the common cases are in a program, and make sure that they go straight through, that they don't get bogged down with all these special case checks. If your main interactive loop has all sorts of checks in it, then your program is going to be slower than somebody else's.

INTERVIEWER: When you come up with an idea to do the greatest word processor in the world, what do you do? How do you design it? Do you look at all the other word processors that are out there?

GATES: Yeah, feature-wise, you look at all the other word processors out there and you say, "Does anybody do kerning on the screen, or do they show you exactly what the printed page is going to look like? How fast do they do it?" Usually, at the top of the product line, there's somebody who took very, very expensive hardware and used brute force to solve a problem. We can't do that; the computer we're working with has very finite speed. A lot of things we do have been done on more powerful computers; we're just trying to make it reasonable to do them on the machine that there are millions of.

You can do an amazing number of tricks inside of a product. You build up your feature list at the same time you're trying to answer the question, "Why will our algorithms be better than anybody else's?" Features are kind of *crummy* in a way, because the more

features you have, the bigger the manual is. And features are only beneficial if people take the time to use them, whereas speed—if you can print the pages faster, or show it on the screen faster, or recall it faster—that's worth an incredible amount. If you can give the users a few simple commands and make the program efficient enough to do what they want with those few commands, then you're much better off. One sign of very good programs is that even internally they follow that philosophy of simplicity. If they want to do something complex, they call the code with simple operations internally, rather than doing the complex operation from scratch.

INTERVIEWER: How important is the end user? How do you know what the database manager out there really needs or wants in his database or in his spreadsheet?

GATES: Well, some programmers don't pretend to have any intuition for exactly what end users want, and they're still world-class programmers. But that knowledge of the market is important, especially in the applications group, so we have full-time people who just show customers the code, or look at other specifications, and things of that nature. When Microsoft first started, we did only systems programs. We knew what programmers wanted, because we're programmers. So we wrote BASIC.

INTERVIEWER: What was the most innovative aspect of BASIC?

GATES: The way we allowed the person to get at the full power of the machine. We put in PEEK and POKE, where you can read and write the machine status. We put in tracing routines called TRON and TROFF: We still let the users, even though they were up in this high-level language, get at all the crazy little things they might want to add

onto the machine. And they can understand how memory is used without BASIC. We let them feel like they are in control of their machines. To fit BASIC into 4K, we used a scheme called a single-representation interpreter. It was a very good choice. I'd never seen an interpreter done that way before. It was kind of risky to do it that way, but I felt incredible confidence in the scheme. I'd run it through in my head and felt good about it.

INTERVIEWER: When you were writing it, did you have any idea it would be as successful as it was?

GATES: No, no way, Paul Allen had brought me the magazine with the Altair on it, and we thought, "Geez, we'd better get going, because we know these machines are going to be popular." And that's when I stopped going to classes and we just worked around the clock. The initial program was written in about three and a half weeks. We ended up spending about eight weeks before I had it fully polished the way that I really liked it. And then I later went back and rewrote it.

No great programmer is sitting there saying, "I'm going to make a bunch of money," or, "I'm going to sell a hundred thousand copies." Because that kind of thought gives you no guidance about the problems. A great programmer is thinking: Should I rewrite this whole subroutine so that four people, instead of three, could call it? Should I make this program 10 percent faster? Should I really think through what the common case in here is so I know how to order this check? If you're a great programmer, you make all the routines depend on each other, so little mistakes can really hurt you. That's why you have to have such fine judgment, and be willing to back up and change things.

INTERVIEWER: When more than one person works on a program, how can you make sure all the different elements are working together properly?

GATES: Well, first of all, the programming team has got to be made up of people who respect each other, because the work is really intimate; it's like being in the same play together. So much judgment and creativity goes into a programming project. Some of the great programmers can't work on teams; they just like to work on their own. But I think there's an element of greatness that comes in learning how to work with other people and teach them. I really get satisfaction from somebody else on the team becoming a great programmer. Not quite as much as I do from writing the program myself, but that is really a positive event. The way I make someone else a great programmer is to sit and talk with him a lot, and I show him my code. In a team project, you make the code everybody's code.

INTERVIEWER: Did this kind of process just evolve here or was it through deliberate implementation?

GATES: Before Paul and I started the company, we had been involved in some large-scale software projects that were real disasters. They just kept pouring people in, and nobody really knew how they were going to stabilize the project. We swore to ourselves that we would do better. So the idea of spending a lot of time on structuring groups has always been very important.

The best ideas are the obvious ones: Keep the group small, make sure everybody in the group is super smart, give them great tools, and have a common terminology so everybody can communicate very effectively. And outside the small groups, have some very experienced senior people around who can give

advice on problems. There is an amazing commonality in the types of difficulties you run into. In design reviews, I really enjoy being able to provide advice, based on programs that I have done.

INTERVIEWER: You mentioned mathematics earlier. What's the relationship between computer science and mathematics?

GATES: Math really affects computer science. Most great programmers have some mathematical background, because it helps to have studied the purity of proving theorems, where you don't make soft statements, you only make precise statements. In mathematics, you develop complete characterizations, and you have to combine theorems in very non-obvious ways. You often try to prove that a problem can be solved in less time. Math relates very directly to programming, maybe more so in my mind than in other people's minds, because that's the angle that I came from. I think there's a very natural relationship between the two.

INTERVIEWER: Does accumulating experience through the years necessarily make programming easier?

GATES: No. I think after the first three or four years, it's pretty well cast in concrete whether you're a good programmer or not. After a few years, you may know more about managing large projects and personalities, but after three or four years, it's clear what you're going to be. There's no one at Microsoft who was just kind of mediocre for a couple of years, and then just out of the blue started optimizing everything in sight. I can talk to somebody about a program that he's written and know right away whether he's really a good programmer. If he's really good, he'll have everything at the tip of his tongue.

It's like people who play chess.

When you're really into playing chess, it's easy to memorize every move in ten chess games, because you're involved in it. Other people look at that recall in chess players, or in programmers, and they think it's like some freak show. But it's completely natural. To this day, I can go to the blackboard and write out huge slabs of source code from the Microsoft BASIC that I wrote ten years ago.

INTERVIEWER: What does it feel like when you are programming?

GATES: When I compile something and it starts computing the right results, I really feel great. I'm not kidding, there is some emotion in all great things, and this is no exception. It's tempting to just start typing the code in, but there's nothing worse than typing in a routine just to get the results, and then realizing that all the hard stuff still has to be written. Because if that's true, you're going to have to change what you've already done. I like to wait and really build the foundation before I let myself have the enjoyment of coding it and seeing it run. It's like saving the best thing on my plate for last.

INTERVIEWER: Do you see a difference between the way young programmers go about programming and the way older programmers do?

GATES: Programmers just starting out today never had to squeeze, so it's a little harder for them to get the right religion because they always think of resources as being immediately available. Ten years ago every programmer ran into resource limitations, so the older programmers are always thinking about those things.

Programming takes an incredible amount of energy, so most programmers are fairly young. And that can be a problem, because

programming requires so much discipline. When you're young, your goals aren't as stable; you may get distracted by one thing or another. Young programmers should stick with it, though, and they'll get better. I think I improved dramatically as a programmer between 1975 and 1980. In '75, I would have said, "Hey, watch out, I can do anything." I really thought I could, because I had read so much code, and I never found a piece of code that I couldn't read very quickly. I still think that one of the finest tests of programming ability is to hand the programmer about 30 pages of code and see how quickly he can read through and understand it.

INTERVIEWER: Do you think that's a talent?

GATES: It's a talent, you bet. It's kind of like pure I.Q. You have to just concentrate on the code and relate back to programs you've written. A lot of people would say, "I want days and days to read this." A really good programmer would say, "Let me take that home with me. I'll just spend an hour tonight and go through the whole thing." The difference of ability there is vast.

INTERVIEWER: Is studying computer science the best way to prepare to be a programmer?

GATES: No, the best way to prepare is to write programs, and to study great programs that other people have written. In my case, I went to the garbage cans at the Computer Science Center and I fished out listings of their operating system.

You've got to be willing to read other people's code, then write your own, then have other people review your code. You've got to want to be in this incredible feedback loop where you get the world-class people to tell you what you're doing wrong. You can't let little idio-

syncrasies get in the way of getting that feedback. Some of the world-class people will harp on some purely idiosyncratic detail, like how you comment the program. You have to cut through all that stuff, because in a way, they're trying to create programmers in their own image; they're trying to get you to do arbitrary things the same way they do. And it may not relate to pure quality issues.

If you ever talk to a great programmer, you'll find he knows his tools like an artist knows his paintbrushes. It's amazing to see how much great programmers have in common in the way they developed - how they got their feedback, and how they developed such a refined sense of discipline about what's sloppy and what's not sloppy. When you get those people to look at a certain piece of code, you get a very, very common reaction.

INTERVIEWER: Has anyone in particular influenced you in the way you write code?

GATES: Everybody who wrote the PDP operating system influenced me. And John Norton from TRW, who wrote memos about people's code--I'd never seen anything like that before. I started trying to do that with other people's code myself.

There's been a mixing of ideas between Paul Allen and me, because so much of the programming we've done, we've done together. It is nice to have somebody who's up to speed to talk to when you're debugging code or you aren't sure about some particular trade-off. In a sense, it's a way of taking a break, relieving the intensity without having to switch topics, just going in and discussing it with somebody. In the creative process, it's good to have the pressure off a little bit and yet still have your attention focused. Paul and I learned how to work

together in an effective way. You don't find too many partnerships like that. He's had a huge influence on me. And then, in the Microsoft period, Charles Simonyi and some of the other people here have influenced me, too.

INTERVIEWER: Microsoft is a broadly based company and the industry as a whole is changing rapidly. How do you keep up with everything?

GATES: Well, I don't keep up with everything. I'm working with the top people at IBM, at Apple, at DEC, and in Japan. I have to know what's going to happen; I can't waste a lot of time guessing. When I fly somewhere with somebody from Microsoft, we talk about what's happening. And the E-Mail system at Microsoft is an efficient vehicle that helps me keep up.

One way to keep up is to use personal computers, and to make sure that I've read the manuals and used the top ten software products. Those products don't change so often that I can't be very familiar with each of them. If you really give a darn about personal computing, you will have used every one, and know something about them and think about how you could do better than those packages.

In a sense, personal computers have become simpler. Now we have just the two architectures: the PC and the Mac. Back in the good old days we had thirty or forty different machines that were totally incompatible, and there were a whole bunch of languages that people were messing around with. Because we've brought millions and millions of people in, we've had to make it more homogeneous, more standardized, so that they can get some sense of what's going on. A lot of what goes on in the industry doesn't really advance the state of the art. I like to concentrate on networking or graphics, because

what we're going to do there might have something to do with the state of the art, instead of concentrating on did this retail chain go bankrupt, did this guy bribe that guy, or did this company give this guy enough stock? Who cares? The really smart people here are focused on their areas; they bring me anything they think is of significance, any project that's going to have real impact.

INTERVIEWER: Where do you see Microsoft in ten years?

GATES: Our goals are very simple. Were going to create the software that puts a computer on every desk and in every home. I don't know if that'll take ten years—that's not my expertise, guessing those exact time frames. Microsoft also wants to participate in helping

to make sure those machines are good machines, building the system software into them, and then doing a lot of the important applications they'll use.

And even though there'll be more and more machines, our present thinking is that we won't have to increase the size of our development groups, because we'll simply be making programs that sell in larger quantities. We can get a very large amount of software revenue and still keep the company not dramatically larger than what we have today. That means we can know everybody and talk and share tools and maintain a high level of quality.

One of the new areas we're focusing on at Microsoft is compact-

disk applications. CD ROM is the technology we're going to use to get personal computers into the home.

INTERVIEWER: Do you ever wish you were back programming on your own again?

GATES: Oh, sure, absolutely. Then you control everything. There's no compromise. Every line is yours and you feel good about every line. It's kind of selfish, but it's like being allowed to do pure mathematics, and yet you have the feedback of making something really work. I sometimes envy my colleagues who get to focus just on the program they're writing.

Excerpted from the book *Programmers at Work* by Susan Lammers. Reprinted with permission of Microsoft Press.

BACK ISSUES OF Algorithm

An increasing number of readers, especially new subscribers, have been inquiring about back issues. What was fun then is fun now!

Algorithm 1.1: The Mandelbrot Set, Hailstone Numbers, Artificial Life

Algorithm 1.2: IFS fractals, Strange Attractors, The Oracle Machine

Algorithm 1.3: Lightning Lines, Cellular Automata, The Volksrobot

Algorithm 1.4: Foxes and Hares, Digital Gardens, Fractal Forms, Neural Nets

Algorithm 1.5: Computer Dunes, Mathematical Black Holes, Growing fonts

Algorithm 1.6: Programmed Collections, Screensavers Contest, Moiré machines, LEGO and Logo

To order back issues, send a check or money order along with a copy of the form on the right or a reasonable approximation thereof. We will send you, post-haste, the reading and recreation you have missed.

Mail your order to:

Algorithm

P.O. Box 29237, Westmount Postal Outlet
785 Wonderland Road
London, Ontario, CANADA
N6K 1M6

Please send me the following back issues of *Algorithm*.

issue	number of copies	cost
1.1	_____	_____
1.2	_____	_____
1.3	_____	_____
1.4	_____	_____
1.5	_____	_____
1.6	_____	_____

I have enclosed total: _____ in payment.

NAME: _____

ADDRESS: _____

} enter \$5.00 in
U.S. funds for
each copy
ordered.

ALGOFICTION

by A.K. Dewdney

Programming Roger

Roger (so Chip called his new purchase) stood resplendent in the middle of the room. It had taken Chip the better part of a day to unpack the personal robot, then to snap on the arms, legs and head, to connect the control lines between parts and to read the introductory section of the enormous manual.

"To record the name, set the automatic functions switch (see Figure 1.7) to AUX, press 0015 on the chestboard, and state the name as clearly as you can ..."

According to the manual, to test the name (at any time) he had only to speak it clearly.

"Roger?"

"Hello. Thank you for giving me the name Roger and congratulations on completing the name entry. It shows that you are learning how to program me already. Of course, robot programming is a deep and complex art. Moravec Robots strongly recommends Microsoft Behave or the Lotus Fun To Be package. Until appropriate behavior programs are installed, I am capable only of these simple, voice-activated responses."

Chip consulted the manual:

"For special-purpose robotic roles, it may be necessary to program your Robot directly in MacroB, the flexible, all-purpose macro-language that enables you to write any behavior program you like."

As an experienced programmer, Chip felt he could write his own custom robot package, one more suited to his needs. Among these needs was a private yearning for companionship. Chip had never been able to get close to other humans, perhaps because of his obsession with machines. Over supper, he read about MacroB, hastily jotting down various commands. By the time he had

bolted the last of the Irish stew, he already had a test program in mind.

FLUSH

Find stairs

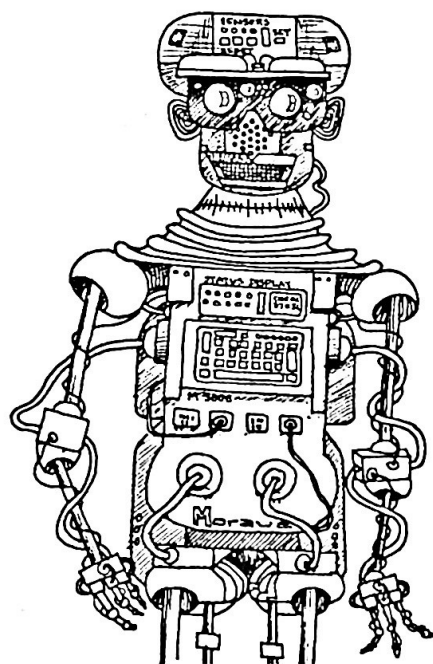
Ascend

First door: bathroom

enter, find toilet: round, white, on floor

push down on handle

return



He typed the program into his computer, then called Roger.

"Roger, come here."

Roger trundled with a scuffling, bumping noise to the study and stopped before his desk.

Chip plugged the transfer cable into Roger's chest and pressed the enter key on his own keyboard. Roger's eyes seemed to flicker briefly.

"Roger?"

"Yes?"

"Flush."

Roger wheeled about and left the office in search of the stairs. His master followed to peek around the corner. Gingerly, Roger lifted his legwheels, one after the other, until

he disappeared from view. There were some creaking noises, the sound of the bathroom scales being pushed out of the way. Then the flush. Chip grinned to himself and went back to sit before his computer, waiting.

Presently, Roger skated into the room and halted once again before him.

"Flush is completed."

"This is truly excellent," thought Chip. "I'll turn Roger into a thing of wonder."

While Chip slept, Roger stood completely immobile on the living room floor, moonlight dappling the cowl over his multi-spectrum eyes.

On day two, Chip experimented with more complicated programs. By evening, Roger could translate the commands "Turn on the TV," "Bring me the mail," and "Get a can from the cupboard" into meaningful actions that greatly pleased Chip to watch. It gave him the feeling that Roger actually had intelligence and that, as he went about his business, he was thinking of other things as a true companion would.

By the end of day three, Roger could "Mow the lawn" and "Bring up the laundry." The lawn-mowing program had taken considerable ingenuity of Chip's part. No matter where he started Roger, he seemed to mow the lawn successfully. True, he didn't mow under the picnic table, but that would come later.

Things started to get complicated when Chip decided that Roger should be active all the time. That way, he would seem more like a real companion. Luckily, there was a way of stringing macros together:

Mow lawn

Bring up laundry

Repeat

get a can from the cupboard

Until hear: "stop that nonsense"

Turn on TV

Wait

Chip whistled happily to himself as he listened to the ancient gas mower roar over the grass. The lawn did not need cutting since Roger had mowed it 12 times already in the course of having his mowing program debugged. In fact the lawn looked rather like a football game had just taken place on it. But that did not bother Chip. Roger was his companion.

When the mower's roar died away, Roger eased himself up the back porch stairs, through the kitchen and down into the basement to bring up the laundry. Chip had carefully left a basket there for Roger to carry up. Now came the fun part: Roger began to remove cans from the cupboard, placing each on the kitchen table, then going back to get another. It was Chip's first try at a loop.

"What are you doing, Roger? Stop that nonsense!"

Roger stopped removing cans from the cupboard. He paused to consider Chip blankly a moment, then wheeled into the living room to turn on the TV set. He stood beside the set, waiting.

In spite of the arbitrary act of removing cans from the cupboard, it was clear that loops were going to work well. It fired Chip's imagination. This was only the beginning. The key to his companion would lie, surely, in the use of loops. There would be loops for every behavior, and loops within loops. He pondered Roger's faceplate.

Then he sat down to write a new program:

YOU'RE MY FRIEND

ongoing: *while not speaking*

hum: *Dukas: the Sorcerer's*

Apprentice

While date > 1/1/91

if time > 0700

make coffee

go to bedroom

say: "wake up Chip old boy"

go to kitchen
wait until: see Chip
say "How about some java, Chip"
pour coffee
go to bathroom
fill tub: hot
say "Hey Chip, your bath is ready"
repeat:
get can from cupboard
until: no cans in cupboard
repeat:
put can in cupboard
until: no cans on table
mow lawn
take mower to house next door
knock on front door
say "I am Roger. I would like to mow your lawn"
mow lawn
return to this house
get can from cupboard
open can
get saucepan
pour can into saucepan
put saucepan on stove
turn on stove
say "Breakfast is almost ready, Chip."
 ...

Chip sat back to examine the half-finished program. In spite of the somewhat arbitrary nature of certain actions, the program had already taken on a certain elegance in Chip's mind. It even had humor. And the idea of mowing the neighbor's lawn had been a stroke of genius. It was not easy to earn Brownie points in this upscale neighborhood.

If programming Roger was half as easy as it seemed, there would be no end to the improvements he could make. Chip especially looked forward to breakfast, being perfectly happy to eat whatever Roger might surprise him with: Beans, Roger? How nice of you to think of that!

He stayed up late to finish all the special procedures. Then he plugged the patch cord into Roger's

chest and loaded YOU'RE MY FRIEND. Just before midnight, long after Chip normally went to bed, he gave Roger the command:

"Roger. You're My Friend."

Chip stumbled upstairs. Before falling into the heavy slumber known only to those who work very hard, he smiled in the darkness. Then he fell fast asleep. He slept for approximately ten minutes.

"Wake up, Chip old boy!"

Something had gone wrong. It was not dawn, but pitch black.

A grey shape squeaked and rumbled out of the room and made its way down the hall, humming The Sorcerer's Apprentice like someone breathing helium. Chip threw on his robe to follow the machine.

"Wait! You're not supposed to do this until seven in the morning!"

In the kitchen, Roger waited quietly beside the coffee-maker. As soon as Chip snapped on the light, Roger picked up the coffee pot, still humming.

"Dum DUM, da-da-da-dum DEE Dum ..."

Chip ran to the living room to search for the manual. He recalled something about "emergency over-ride."

Then came the sound of liquid being poured carefully onto the kitchen floor.

"How about some Java, Chip?"

Chip's mind was not ready for this. Numbly, he noted the bug in the coffee-serving procedure, even as Roger passed through the living room on his way upstairs. Someone had neglected to index the emergency over-ride in the manual. Chip would have to search the monstrous tome page by page. Upstairs, the taps came on.

About the time that Chip began to ponder how any designer in his or her right mind would build an over-ride switch that had to be enabled before start-up, water

began to drip through the ceiling.

"Oh God!"

Roger appeared, his wheeled feet frosted with soap suds.

"Hey Chip. Your bath is ready."

He passed on into the kitchen.

Chip ran upstairs.

The bathtub was running over. The floor was a sea of bubble-bath (a token, in Chip's program, of Roger's desire to please him). Chip plunged his arm into the scalding water to pull the plug. Why hadn't Roger seen that the tub was full? Did the bubble-bath obscure his view of the water?

The flaws in the program were taking on nightmare dimensions. Chip raced back downstairs, trying to recall what the program would do next. Roger was removing cans from the cupboard. He hummed contentedly as Chip remembered something about the lawn.

"Dum DUM, da-da-da-dum DEE Dum ..."

Chip raced outside into the darkness, wheeled the lawn mower behind some bushes, then brought the small gas can into the house as an extra precaution. He set it on the table.

Then he desperately tried some vocal hacking:

"Roger! Stop! Halt! Quit! Cease! End! Terminate!"

None of the commands had the slightest effect on Roger. Chip thought of phoning the police. But what if they had to shoot Roger? He found the idea unbearable.

Then he thought of the Moravec customer hotline. It was inside the front cover of the manual. He dialed the number, mumbling a half-forgotten prayer. Roger had begun to replace the cans in the cupboard. Among these was the lawn mower gas can.

"Robot hotline. How may we help?"

"I've got a robot I can't stop. I need a voice command that will

stop it right away."

"That would be customer relations."

"No, you idiot, it would not be customer relations"!

Roger trundled out into the night, his infra-red picking up the yard, the trees and the shed quite nicely. The lawn mower, which still glowed from its earlier use that day, was parked among some bushes, a fact that puzzled Roger not in the slightest.

The ancient gasoline engine split the pre-dawn darkness.

"My God, can't you hurry?"

"Let me connect you with our programming consultant."

"That's better."

By infra-red, the lawn looked much different to Roger. In fact, there was nothing to mow, by all appearances. He pushed the mower, still roaring, out the driveway and down the street to the neighbor's house.

Chief Justice Richardson, a grumpy man when in the best of humors, had of course been awakened by the lawn mower. The knock on his front door brought him downstairs in a murderous rage. But when he saw the gleaming concoction of steel and plastic in front of him, he breathed a sigh of relief. He was clearly having an extremely bad dream.

"I am Roger. I would like to mow your lawn."

"Certainly."

Judge Richardson shut the door and frowned to himself. It was unusual to have so much control in a dream. He decided to phone the police, just in case.

By the time the police came, Roger had concluded not only that the judge had no lawn, but that none of the neighbors did either. That meant he could go home. His humming could hardly be heard above the sound of the throbbing mower. Lights were on all over the

street. Two squad cars pulled up in front of the judge's house.

Chip hung up the phone, turned off all the lights and crept upstairs. He would pretend he had nothing to do with Roger. He lay quietly in the darkness of his bedroom, listening to the disturbance. Fragments of speech drifted up to his bedroom window. Shouts mostly.

"... One of those personal robots ..."

"... grab the lawn mower ..."

The lawn mower engine suddenly quit.

"It's heading back to you ..."

There were some scuffling sounds. Then Roger came in through the back door of the house. Chip buried his head under the pillow, but it was no use. Someone was knocking at the front door.

Wearily, Chip made his way down the stairs to find a uniformed officer waiting under the porch light.

"I'm sorry to trouble you, sir, but we have reason to believe that someone's personal robot has entered your house."

"You don't say! A robot?"

The officer looked over Chip's shoulder, an expression of concern on his face.

"Excuse me, sir, but I think you have a fire."

"Oh, my God."

Chip ran to the kitchen. The stove was a mass of flame that roared to the ceiling, illuminating in its hellish glare Roger's impassive face and the gas can sitting beside the stove. Roger's plastic speaker grill was sagging, giving him a hideous grin. Inside, his delicate silicon circuits were about to shut down permanently. He just had time to stop humming, adjust the pot of flaming gasoline and turn cheerfully to his mentor.

"Breakfast is almost ready, Chip!"

MICRO SCOPE

by David Geoffrey

The response to our request for software to review has left us a bit breathless but very pleased. So much has come in that we won't be short of ideas for several issues. Don't let this stop you, however! Please, keep those submissions coming in.

One problem, however, is that we're receiving BIG submissions. It is taking us several hours per program to get familiar enough to review them. Isn't there some smaller stuff out there? Don't be put off just because your program doesn't fill a 360k floppy.

We are going to attempt to stay with one particular computer type per column. All of this issue's software runs on IBM PC or compatible machines and does best with VGA graphics. Next issue we will examine some of the Macintosh software that has come in.

GRASS ISN'T ALWAYS GREEN

There was grass growing on my screen. Long wiry twitch grass, we used to call it. It was growing on my computer screen. Now it was topped with some dirty gray snow. I wasn't quite sure what to do next so I just watched. Moments later my grass had grown into a rainbow variety that I'd never seen on a lawn before.

I was watching the output of an inventive little time waster called FR2. Not an imaginative name perhaps, but easier to spell than "Ballistic Aggregation Techniques for Generating Fractals." Written by Edward Richards, the program grows many different fractals by randomly "dropping" points from the top of the screen onto a base line at the bottom of the screen; that is the Ballistic part. As the points drop onto, or near, other already dropped points, they aggregate or clump together.

A simple menu-driven interface

allows the user to specify some of the parameters that control the program. One can choose how the points to be dropped are distributed across the top of the screen and how those points will stick when they meet others. Widely different results will be obtained by varying this; your points can slide down the sides of those it meets, like sand down a sandpile, or stick when they meet, creating branches of trees or blades of grass. The resemblance to grass, trees, bushes, and coral is quite amazing.

To make it a truly marvellous time-waster, it is possible to slow down the program. Normally, you see the points at the top of the screen and then again as they aggregate at the bottom. A "slow" option behind one of the menus allows you to watch those points as they drop. If you choose the random walk falling pattern, you could be mesmerized for weeks! Most amusing, when those staggering points finally meet another point, they seem to do a little dance of joy.

The program runs on any of CGA, EGA, or VGA monitors. Under CGA, it works in monochrome for the best resolution. In EGA and VGA it changes the color of the dropped points every so often, resulting in fractals with waving bands of color.

This program is available for \$25.00 (US) from:

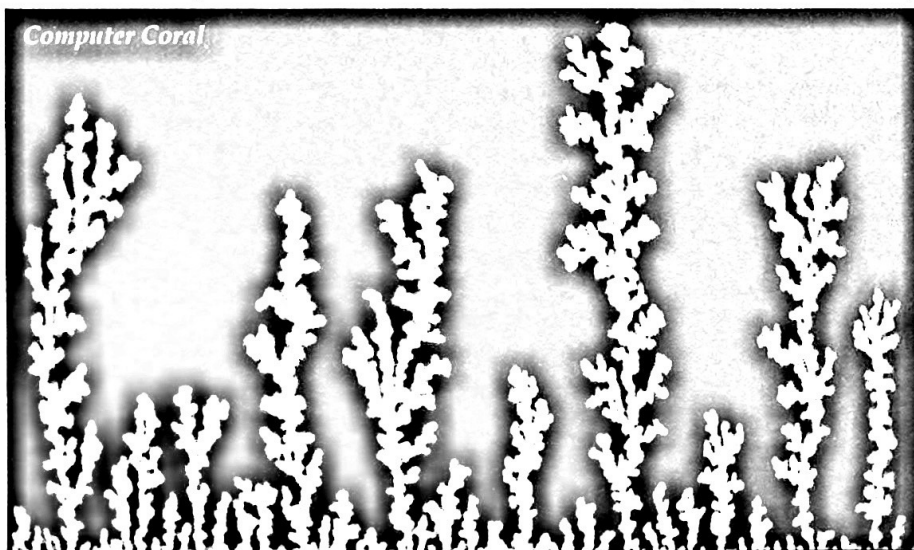
Fractal Recreations
21 Wichard Blvd.
Commack, New York
11725-1702, USA

MANDELBROTS OF FIRE

Exploring the Mandelbrot or Julia sets can be a tedious occupation because of the complexity (pardon the pun) of the math. It is not unusual to wait hours for a good picture to be produced. Michael Freeman has constructed a tool that removes some of that tedium.

Allowing exploration of both the Mandelbrot and Julia sets, VGAMBROT allows the user to select from among several algorithms for the calculation of the set. The best of these methods produces good plots in rather short times. VGAMBROT also allows the user to queue up to eight plots at once; the computer will plot them all and place the output in files for later viewing. This makes an overnight run possible.

I was most impressed by VGAMBROT's ability to animate the Mandelbrot and Julia sets. With correctly selected color maps, the set catches fire. Tongues of flame leap



from the edges of the set or run down into the valleys that cut away from the set. The technique is simple: rotate the color map. The result is spectacular.

I was a bit disappointed by the default color selections, finding them washed out. Fortunately, you have complete control over the color selection and can change the colors to suit personal taste.

VGAMBROT's mouse interface is somewhat cumbersome. I found too many times that I was having to go to the keyboard to answer questions which menu selections could have handled. In many respects, the non-mouse interface was easier to use, even though several features cannot be invoked without a mouse.

VGAMBROT is not an easy program to learn. It requires a serious investment of time to become familiar with its capabilities. The 13-page user's manual, while very complete, is structured more as a reference manual than as a guide. I found it necessary to experiment with the various functions in order to understand what the manual meant.

VGAMBROT is a cumbersome but extremely powerful tool. It requires at least an EGA graphics monitor and really shines on VGA. It runs best with a mouse and a math co-processor. It is available for \$30.00 from:

Michael Freeman
4777 Hoskins Road North
Vancouver, B.C. Canada,
V7K 2R3

IF YOU COULD SEE WHAT I HEAR

I sat and watched a Bach Fugue (#4, in C sharp minor from the Well Tempered Klavier, Book I) march across my television set. Like a flock of mismatched birds, stately flamingos mixed with frantic

sparrows, the music paraded across the screen, seemingly driven by the music coming from the speakers. In truth, it was; I was watching the output of Stephen Malinowski's Music Animation Machine, a tool for the exploration of musical structure. Just minutes later a Chopin Etude would take me for a roller coaster ride as my eyes tried to follow what my ears were hearing.

When a musician talks about the structure of a piece of music, the non-musician tends to treat him or her much like non-computing people treat someone who always talks about bits and bytes: with disdain. To most people the concept of voices in a fugue crossing each other means absolutely nothing; they don't have the underlying know-ledge needed to understand.

Stephen Malinowski has created a tool which can break this musical barrier. The Music Animation Machine, running on an IBM PC, or compatible, listens on a standard MIDI (Musical Instrument Digital Interface) port and produces a VHS videotape of the music. He is currently making a demonstration videotape available.

The visual output of the program resembles a player piano roll as it scrolls from left to right across the screen. Lines of color, where the length of the line is determined by the length of the note and the vertical position on the screen is determined by the pitch, represent the notes in a piece of music. As they scroll across the screen, elements of the structure of the piece of music become immediately evident. I've run the tape for several music teachers and, without exception, they have been excited about using this tape for teaching music. They see it as useful for teaching beginning music students what up and down means musically, for demonstrating rhythm,

for analysis of a composition, and much more.

Stephen Malinowski has built a tape that belongs in every music educator's library. If you teach music or know someone who does, this would make an excellent gift. I do hope that the Music Animation Machine software is going to be made available soon.

To get your copy of this VHS videotape, send \$25.00 (US) to:

Music Animation Machine
1850 Arch Street #5
Berkeley, CA, 94709, USA

MICRO SCOPE SUBMISSION GUIDELINES

If you have a piece of software that you would like others to know about, why not send us a copy? There are only a few guidelines that we will insist on:

- 1) Please document your programs fully. The more complex your software, the more documentation there must be.
- 2) Please send us your software on 5.25" or 3.5" diskette. We can very probably run it, no matter what machine it came from.
- 3) Please don't expect us to send back your diskette. We will try real hard to respond to every submitter. If for any reason, we feel that your submission is not acceptable, we will let you know how to correct it.
- 4) Please let us know if you are willing to make your software available to others. We cannot distribute it for you but we will print your address so others can contact you.

All submissions should be sent to the Review Editor in care of Algorithm, 362 Wortley Road, London, Ontario, Canada N6C 3S2.

BULLETIN BOARD

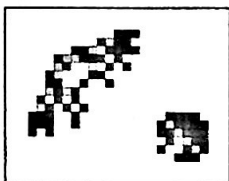
Software, games and publications from individuals and companies.

First appearance of a Bulletin Board ad is free. Subsequent appearances cost \$20.00. Graphics and typesetting of Bulletin Board ads from submitted layouts and art is a free service of Algorithm.



CellMaster

Explore the exciting world of cellular automata. Over 100 models and shapes including Virtual Computer components, and never before seen parametia and jellyfish-like shapes that crawl and reproduce.



Fast! Flexible! Fascinating!
IBM PC, PS/2 (3.5 or 5.25" disks): \$45.00

Sintar Software 206-625-1213
1001 4th Ave., Suite 3200
Seattle, WA 98154
Visa/MC/AmEx accepted

Have you ever seen a function of a complex variable? Would you like to really understand differential operators like *grad*, *div*, or *curl*? How about a peek into the 4th dimension?

Ask about our latest software for PC or Macintosh

Lascaux Graphics 3220 Steuben Ave.
Bronx, NY 10467 USA (212) 654-7429

SUPERMAND

Mandelbrot and Julia sets in high-resolution color. Fast, huge zoom range. Includes 9 ready-made images.

Requires IBM-compatible, DOS 3.0+, EGA or VGA.
80x87 supported but not essential. Specify diskette size & type. \$20 (Aust. or US) surface mail - add \$5 for air.

Guy Cox, Box 206, Wentworth Building, University of Sydney, NSW 2006, Australia.

Computer Go

An international quarterly devoted to all aspects:

- Game-playing algorithms
- Tournament results
- News, letters, and commentary
- Related AI Issues

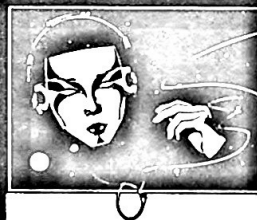
Annual Subscription: USA, US\$18 --Canada, Can\$18
--Elsewhere, \$US18

71 Brixford Crescent, Winnipeg, Manitoba R2N 1E1, Canada

Group of German computer enthusiasts seeks foreign friends to exchange ideas and selfmade programmes.

We use Turbo-Pascal and Basic on IBM-PC, Macintosh SE and Atari ST.

**Jürgen Kellermann • Hedwigstr.
25 W-8901 Herbershofen • Germany**



J. of B.A.S.R.A.

Journal of the British-American Scientific Research Association
Specializing in avant garde science and speculations in science.

A forum for ideas in the sciences.

For information write to: **BASRA**
13 Durwood Place
Madison, NJ 07940

If you have a Commodore 64 or 128

The world of mathematical artistry is within your reach too!

Mandelbrot Set - full control, colour, zoom, high speed.

Cellular Automata - set rule, seed, edge etc.; or auto. Warp speed

Darwin - evolution on-screen in minutes! Scientific Am. May 89

3-D Projection - view math functions & rotate 3-D objects.

Alphametic - high-speed on-screen solution of addition puzzles.

Magic Monkeys - animated, entertaining arithmetic for kids.

\$10 each, or \$30 for all 6 above:

Also \$10 each: **Julia Sets; Affine Transformations**
Ian Adam 4425 West 12th Ave.
Vancouver, BC, Canada V6R 2R3



RECREATIONAL & EDUCATIONAL COMPUTING

REC is the only publication devoted to the playful interaction of computers and 'mathemagic' - from digital delights to strange attractors, from special number classes to computer graphics and fractals. Edited and published by computer columnist and math professor Dr. Michael W. Ecker, REC features programs, challenges, puzzles, program teasers, art, editorial, humor, and much more, all laser-printed. REC supports many computer brands as it has done since inception Jan. 1986. Back issues are available.

To subscribe for one year of 8 issues, send \$24 (US introductory rate only) to REC, Attn: Dr. M. Ecker, 909 Violet Terrace, Clarks Summit, PA 18411, or send \$10 for three sample issues, creditable towards subscription.

EduTech's CHAOS Plus...

Attractors, Fractals, Automata

For Apple II Computers (II+,E,C,GS)

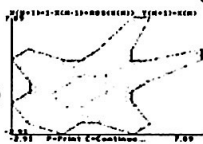
\$75.⁰⁰ US - Money Back Guarantee - Free Catalog



EduTech

1927 Culver Road
Rochester, NY 14609

(716) 482-3151



FRACTAL RECREATIONS

for IBM PC: VGA, EGA, CGA

Generate your own fractals using **BALLISTIC AGGREGATION TECHNIQUES**. Beautiful fractals resemble coral, bushes, trees and mountains. In color on EGA or VGA. Many hours of fun and education. Menus and Help plus hot keys provide ease of use. Introductory offer: \$25.00

Contact: **FRACTAL RECREATIONS**
21 Wichard Blvd.
Commack, NY 11725-1706



(516) 543-3850

What is the APL Programming Language?

APL is a quick and efficient tool for exploratory numerical analyses of *Algorithm* puzzles/articles.

Call the BBS\APL 301-384-3672 3/12/2400 baud, 24 hrs. Many downloadable files (IBM/DOS, not Apple), including 2 free APL interpreters.

Quelques fiches et menus en français.

New Release

WHAM-DUZLE Graphical Diversion

Experimental Design Kit with source code.

Requires 5.25" PC Compatible with MicroSoft or GWBASIC

Color Monitor optional.

Non-Objective Minimalist Art by rapid tessellation of extended character set.

Send \$30 plus tax check or money order:

Christopher Computer, 28 Anderson St.,
Boston, MA 02114-3648

Ami-FX, Amiga Fractal Exchange

Freely Distributable Magazine of Animations,
Art and Software.

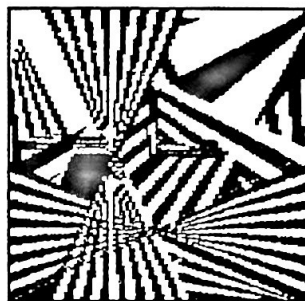
Send Disk and Postage to:

Cade Roux

Gonville and Caius College

Cambridge CB2 1TA

ENGLAND



Hallucinations™ 2.0

Screen Pattern Generator

For IBM & Compatibles

Shareware Disk; \$5; Reg. \$20.

Write for free catalog

Polymath Systems

P.O. Box 795

Berkeley, CA 94701

Expert Systems on your PC

IBIS by

Intelligence Manufacturing, Co.

Inexpensive, Easy, and Powerful

Educational Version-Shareware\$40.00

Development System\$250.00

Contact: **D.E.M. Software**
P.O.Box 456 Woodbine NJ
08270 USA (609) 861-5181

Fractal Explorer®

For IBM® PCs and Compatibles

Only one fractal program offers:

- Almost unlimited zooming capability (10^{72})
- Full screen and quadrant displays • High-speed image generation - up to 100 times faster than the competition
- Edge detection algorithm • Professionally designed user interface • Support for many high-res VGA cards (up to 1024 x 768) in both 16 and 256 color modes.

Send \$29.95 US (check or money order) to:

Andromeda Research, Inc., 6441 Enterprise Lane,
Madison, WI 53719 VISA or Mastercard accepted

PUZLEPACK - \$20.00 US

Solve jumbled words, anagrams, crosswords.

Find all words, any length, in a set of letters.

Fill in "wild card" blanks, fixed or floating.

96,000-word RAM dictionary, amazing speed!

Requires 640K IBM PC compatible or PCjr.

PERMTABL - \$15.00

Permutation tables in unique cyclic form. Up to 16 elements, including duplicates. Sets which generate more than 3 pages of cycles are skipped. 130-page printable file plus generating source (BASIC).

Specify 5 1/4" or 3 1/2" floppies. Check or M.O.

RECROOM RECWARE

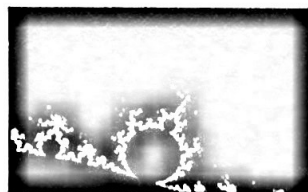
P.O. Box 307, Pacific Grove, CA 93950



FractalMagic

"Infinite... Neither of us (chief engineer of the GALILEO Jupiter probe and I) may ever be heard from again." - Arthur C. Clarke

"It's Glorious! ... I love it!" - Jerry Pournelle, BYTE



Fast! Flexible! Colorful!

IBM PC, PS/2 (3.5 or 5.25" disks),

Mac 512KE, +, SE's, II's: \$35.00

Sintar Software 206-625-1213

1001 4th Ave., Suite 3200

Seattle, WA 98154

Visa/MC/AmEx accepted

COPY -

THIS PAGE

WRITE

US A LETTER

Moving? Have a product or service to distribute? Wish a friend to subscribe? Want to suggest a topic? Fill in the appropriate space below and mail the copy to us at:

Algorithm

P.O. Box 29237 Westmount Postal Outlet
785 Wonderland Road South
London, Ontario, Canada N6K 1M6

Name	
Old address	New address
effective date:	

If you're moving, let us know by filling out the space on the left. This way you can ensure an uninterrupted flow of fascinating projects and engaging articles.

Perhaps you know someone who would like to subscribe to Algorithm, the Personal Programming Newsletter. If so, copy this page and fill in the space on the right.

PLEASE send me 6 issues of Algorithm for the introductory price of **\$29.95**

NAME: _____

STREET: _____

CITY: _____

STATE: _____

COUNTRY: _____ ZIP: _____

PLEASE SEND PAYMENT TO:
Algorithm P.O. Box 29237, Westmount Postal Outlet, 785 Wonderland Rd.,
London, Ontario, Canada N6K 1M6
(Cheque or money order No C.O.D.)
Can. Funds \$35.95 • Overseas add \$4.00 Shipping.

If you have a product or service you'd like to make available to our readers, fill out the space to the left with a rough sketch of your ad and we will set it in type. First appearance: FREE! Subsequent appearances: \$20.00/issue.

Is there a topic you'd like to see us cover? Fill out the details in the space below:

Dear Editor: _____

Name: _____ Phone: _____

Address: _____ Date: _____